
LA PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO
SOLO PROGRAMADORES
ÑO VI. 2ª ÉPOCA NÚMERO 63
UNA PUBLICACIÓN DE: REVISTAS PROFESIONALES S.L.
975 Ptas. • 1.280 Esc-Cont • 5,86 € (IVA incluido)
JAVA
SERVLETS
y JDBC
DELPHI
Programación de Threads (II)
MULTIMEDIA (y III)
Creación de un reproductor multimedia en Delphi IV
TAPI: API DE TELEFONÍA (y III)
Finalizando nuestro proyecto
LINUX: DNS A FONDO (y IV)
DNS avanzado
LA TECNOLOGÍA ASP (IV)
Análisis de los componentes ActiveX de servidor y BBDD
JAVA: CRIPTOGRAFÍA (IV)
Los algoritmos de clave pública
VB: SUITE DE INTERNET (V)
La transmisión de ficheros
REDES:
Transferencia de ficheros punto a punto (y VI)
CONTENIDO CD
HoTMetaL PRO 6.0 • WinLinux 2000 Final Beta • ActiveSkin 3.0
ElPack 2.35 • Service Pack Manager 4.2.1 • SimApp 1.03 • SIMCA-P 8.0
HSSpy 1.0 • MonitorIT 1.1 • Norton Ghost 6.0
00063
8 413042 303299

TODOS

LOS MESES

EN TU

QUIOSCO



PARA ESTAR AL DÍA EN SOFTWARE SIN GASTARSE UN EURO EN TELÉFONO

- LOS MEJORES ESPECIALES
- DISEÑO
- INTERNET
- MULTIMEDIA
- UTILIDADES
- EN CASTELLANO
- NEGOCIOS
- LO ÚLTIMO
- Y MUCHO MÁS...

SÓLO PROGRAMADORES

Número 63
SÓLO PROGRAMADORES
es una publicación de
REVISTAS PROFESIONALES S.L.

Editor
Agustín G. Buelta
Director
Javier Amado Buiza
Coordinador Técnico
Eduardo De Riquer Frutos
Coordinadoras de Redacción
Gema Romero Moreno-Manzanaro
Cristina Peña del Pozo
Colaboradores
Constantino Sánchez, Juan Luis Ceada, Jorge Delgado, Javier Sanz, Adolfo Aladro, Enrique de la Lastra, Jordi Agost, Vicente A. Sánchez Werner
Ilustración de portada
Manuel Gómez Allende
Maquetación y Tratamiento de Imagen
F. Risco
Consultas técnicas
atecnica@virtualsw.es

Publicidad
Paloma Seidel
Tel.: (91) 304 78 46
Mariano Sánchez (Barcelona)
Tel.: (93) 322 12 38
Pepín Gallardo (Barcelona)
Tel.: 617 09 36 68

Suscripciones
Rosa Tabares
Tel. (91) 304 87 64 Fax: (91) 327 13 03

Preimpresión
Grebe
Impresión
I.G.Pantone
Distribución
Motorpress Ibérica

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 31-3-2000
Precio en Canarias, Ceuta y Melilla: 938 ptas. sin I.V.A.
Con sobretasa aérea: 975 ptas. sin I.V.A.

EDITORIAL

La programación Web en alza

Java continúa avanzando y aunque ya son muy conocidos los *applets*, no lo son tanto los *servlets*. Éstos, junto con el acceso a datos mediante *JDBC*, son una de las opciones que más espectativas tienen en la actualidad para el desarrollo de sitios *Web* potentes y versátiles. En el tema de portada de este mes y con la finalidad de estar a la última, tratamos todos los aspectos que permiten tanto los *servlets* como el *JDBC* y como no, los dos unidos.

Además continuamos con las secciones de tecnología *ASP*, donde hablamos de los componentes *ActiveX* de servidor y las bases de datos que nos facilitan en gran medida la realización de operaciones a partir de objetos predefinidos. Criptografía, donde describimos los algoritmos de clave pública que revolucionaron los sistemas de criptografía actuales. En la *Suite* de *Internet*, aprenderemos a trabajar, programando en *Visual Basic*, sistemas de recepción de ficheros desde sitios *Web*; y programación de *Threads* en *Delphi*, donde mostramos cómo proteger el acceso a los recursos compartidos y cómo gestionarlos desde varios *threads* mediante los diferentes mecanismos que ofrecen tanto *Delphi* como *Windows*.

Muchas de nuestras series finalizan, lo que indica que ya disponemos de conocimientos sobre los temas: *API* de telefonía, donde finalizamos el proyecto de *Viual Basic* del número anterior. Además de Programación de un reproductor multimedia con *Delphi*, donde mediante un icono en la barra de tareas mostramos opciones posibles que nos evitan recurrir al menú del programa, y en el que también ofrecemos unas nociones sobre el estándar *MCI*.

Como podéis comprobar un número muy completo y con el que todos podemos ampliar nuestros conocimientos en un grupo de campos muy amplios, el mes que viene comenzaremos dos nuevas series que esperamos que sean de vuestro agrado.

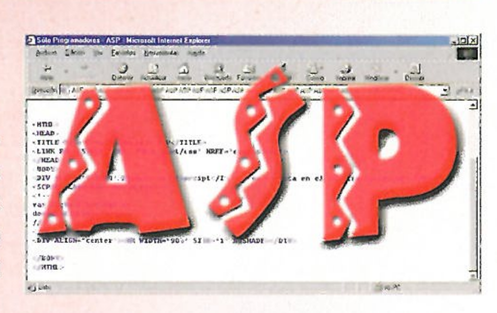
Javier Amado
Director



SÓLO PROGRAMADORES

52 INTERNET

LA TECNOLOGÍA ASP (IV): COMPONENTES ACTIVEX DE SERVIDOR Y BBDD.



Ya nos queda un poco menos para dominar esta nueva tecnología, pues los componentes *ActiveX* de servidor, que veremos en este número, nos proporcionan una forma de hacer ciertas operaciones a partir de objetos predefinidos, lo que facilita enormemente la labor del desarrollador de aplicaciones *Web*. Asimismo estos componentes permiten acceder a todo tipo de fuentes de datos de forma rápida y sencilla.

38 TELECOMUNICACIONES

TAPI: API DE TELEFONÍA (y III)



Finalizamos esta serie sobre telefonía terminando el proyecto de *Visual Basic* que dejamos pendiente en el número anterior. Lo haremos paso a paso a través del formulario que contiene el código principal del programa.

44 LINUX

DNS A FONDO (y IV): DNS AVANZADO



Después de implementar un servidor *DNS* en nuestra máquina *Linux*, en este artículo veremos la relación del *DNS* con el sistema *CIDR* de asignación de direcciones *IP*, las medidas de seguridad que podemos usar y un sistema para realizar modificaciones en las zonas de nuestro servidor de forma dinámica. Con un vistazo rápido a estas cuestiones cerramos esta serie sobre el sistema de nombres de dominio.

58 JAVA

CRIPTOGRAFÍA (IV)

Si los algoritmos de clave privada mostrados en anteriores artículos son muy utilizados y con excelentes resultados, los de clave pública no lo son menos. En este artículo se van a describir los algoritmos de clave pública, que en su momento revolucionaron la criptografía actual, y sus diferentes posibilidades de uso, como el intercambio de claves, firma digital, etc.

66 VISUAL BASIC

SUITE DE INTERNET (V): LA TRANSMISIÓN DE FICHEROS

¿Cómo podemos recibir ficheros de un sitio *Web* concreto? ¿Cómo podemos hacer que nuestra aplicación envíe ficheros de control o sobre su estado a un servidor de *Internet*? ¿Cuáles son los pasos que debemos seguir? ¿Cómo lo podemos realizar con el menor número de controles posible? A éstas y otras preguntas intentaremos responder a través de este artículo.

72 REDES

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (y VI)

La última fase de la implementación del protocolo de Nivel de Enlace es la definición de los procesos que hay que realizar en función del estado en que se encuentre el transmisor o el receptor. Por eso en este artículos contemplaremos la reacción a cada trama recibida, independientemente de ésta y del estado de transmisión o recepción, realizando todas las funciones necesarias de forma transparente al nivel superior.

78 LIBROS

Las últimas novedades editoriales sobre programación y los libros que nos han parecido más interesantes aparecen en estas páginas, con una sencilla reseña que os ayude a conocerlos mejor y a decidir cuál es el que más os conviene.

80 DUDAS TÉCNICAS

Como siempre en esta sección contestamos a todas vuestras dudas. No os preocupéis si os parecen demasiado complejas o demasiado sencillas, siempre podréis acudir a nuestra dirección de e-mail: solop@virtualsw.es. Nosotros intentaremos solucionar vuestros problemas.

SUN MICROSYSTEMS DISTRIBUYE GRATUITAMENTE POR INTERNET MÁS DE UN MILLÓN DE COPIAS DE STAROFFICE

Con tan sólo nueve semanas transcurridas desde que la compañía *Sun Microsystems* comenzara a distribuir el programa de aplicaciones ofimáticas *StarOffice TM* 5.1, ya se ha superado la cifra de un millón de copias de distribución gratuita.



Esta aplicación está disponible, de forma completamente gratuita en todo el mundo, para los sistemas operativos más comunes. Esta *suite* completa de *software* de oficina permite a los usuarios leer y escribir los archivos propietarios, además de posibilitar también el intercambio entre sistemas con otros formatos de archivos de los paquetes de ofimática más populares. Los usuarios se benefician de la *interfaz* única de *StarOffice* para sus herramientas tales como el procesador de textos, hojas de cálculo, diseño gráfico, presentaciones, edición en *HTML*, correo electrónico, además del organizador y las funciones del editor de fórmulas.

StarOffice funciona en los entornos operativos *Solaris (TM), Windows, Linux* y plataformas *OS/2,* y está disponible en ocho idiomas.

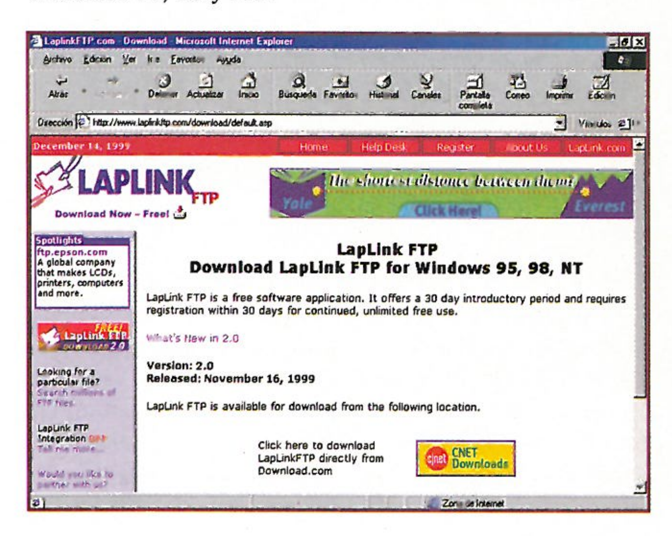
Para más información: **www.sun.com**

YA SE PUEDEN BAJAR ARCHIVOS DE LA RED SIN ESPERAR CON EL NUEVO PAPLINK FTP 2.0

LapLink FTP 2.0 ofrece *LapLink Queue*, que permite a los usuarios transferir archivos mientras continúan navegando por *Internet*. De este modo se seguirá teniendo el control total de los archivos almacenados en espera mientras éstos están siendo transmitidos.

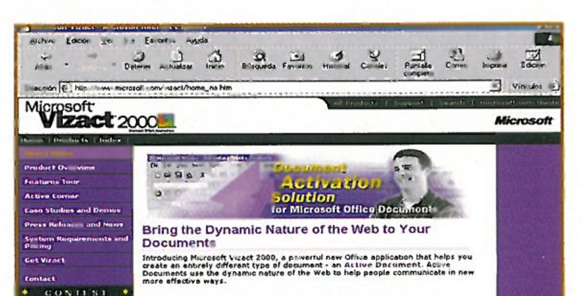
Además de ello, ofrece apoyo de bajada en grupo para que los usuarios puedan planificar bajadas automatizadas para los momentos en que los sitios *Web* experimenten menos tráfico. Gracias a la capacidad de planificación de esta herramienta y su espera en cola se obtendrá un mejor rendimiento a la hora de transferir archivos.

LapLink FTP 2.0 puede ser bajado gratuitamente desde **www.laplink.com** y **www.download.com**. Después de 30 días el usuario tiene que registrarse para continuar utilizando el producto que es compatible con *Windows* 98, 95 y *NT*.



MICROSOFT LANZA VIZACT 2000

Microsoft ha dado a conocer la disponibilidad de *Vizact* 2000, una aplicación que permite a los usuarios comunicarse de forma más eficaz mediante la inclusión en sus documentos *HTML* de las prestaciones dinámicas y atractivas de la *Red*.



La tecnología de esta herramienta permite añadir la funcionalidad multimedia, como la programación, animación e interactividad a sus documentos *Web*, convirtiéndolos de esta forma en "documentos activos". Además, logra reducir la sobrecarga de información y posibilita que los usuarios transmitan mejor su mensaje aprovechando al máximo las últimas tecnologías dinámicas de la *Red*.

Con *Vizact* 2000 se puede controlar la funcionalidad multimedia de los documentos para aumentar la atención hacia los mensajes clave, mejorando la recepción y retención de ideas.

Para más información: **www.microsoft.com/vizact/**

SUN MICROSYSTEMS PRESENTA JAVA MEDIA FRAMEWORK 2.0

La tecnología *Java Media Framework* 2.0 fue creada para la reproducción, sincronización, captura y transmisión de ficheros multimedia a través de la mayoría de sistemas operativos.

Sun e *IBM* crearon el *API* para ayudar a los desarrolladores a evaluar de forma más eficiente las nuevas tecnologías. Dicha *API* proporciona a los desarrolladores de *Java*, capacidades avanzadas para procesar los datos, incluyendo la captura multimedia, junto con soporte de tecnologías como *MP3, Flas, RTP/RTPS, Hotmedia, RMF*, etc.

Esta nueva tecnología incluye nuevos tipos de media como el *QuickTime*, el formato *AVI* de *Microsoft*, y el *MPEG-1*, así como una arquitectura abierta que les proporciona a los usuarios el acceso a los procesos de captura y reproducción.

Para más informarción: **www.sun.com**

NUEVO CHIPSET PARA PC'S DE ALTAS PRESTACIONES

Intel Corporation ha anunciado un nuevo *chipset* que optimiza las prestaciones de los *PC's* basados en el procesador *Intel Pentium III* dirigidos al segmento de mercado de ordenadores de sobremesa de altas prestaciones.

El *chipset Intel* 820 aporta nuevas características a los usuarios de *PC's* incluyendo un *bus* de sistema de procesador más rápido, mayor capacidad de memoria y una mejor funcionalidad gráfica. Además de lo mencionado éste es el primer *chipset* de sobremesa que permite disponer de la tecnología de memoria de altas prestaciones *Direct RDRAM* y proporciona significantes mejoras de prestaciones gráficas mediante el soporte gráfico *AGP 4x*. Gracias a la memoria *Direct RDRAM*, se logra el ancho de banda necesario para obtener prestaciones óptimas. Sumando a esto la tecnología *AGP 4x* aporta un nivel superior de prestaciones gráficas *3D* al permitir incorporar controladores gráficos para acceder a la memoria principal a más de 1 *GB/seg*, el doble de las anteriores plataformas *AGP*. El resultado es una plataforma que proporciona contenidos gráficos y multimedia mucho más ricos en las actuales aplicaciones.

Para más información: **www.intel.com**

Noticias

MICROSOFT ANUNCIA WINDOWS DNA 2000

Microsoft acaba de anunciar la aparición de *Windows DNA* 2000 (*Windows Distributed Internet Architecture*), una plataforma integrada y completa para la construcción y operación de las más modernas aplicaciones *Web* distribuidas, así como la nueva generación de servicios *Web* basados en *Internet*. Se trata de un conjunto de sofisticados servicios *Web* más completos, personalizados y activos que pueden enlazar directamente aplicaciones, servicios y dispositivos utilizando *Internet*.

Windows DNA 2000 proporciona un completo conjunto de servicios de desarrollo y adaptación de sofisticadas aplicaciones. Las herramientas de alta productividad soportan múltiples lenguajes de programación y grados de conocimiento que les hace accesibles por el más amplio conjunto de desarrolladores del mercado.

Para más información: **www.microsoft.com/spain**

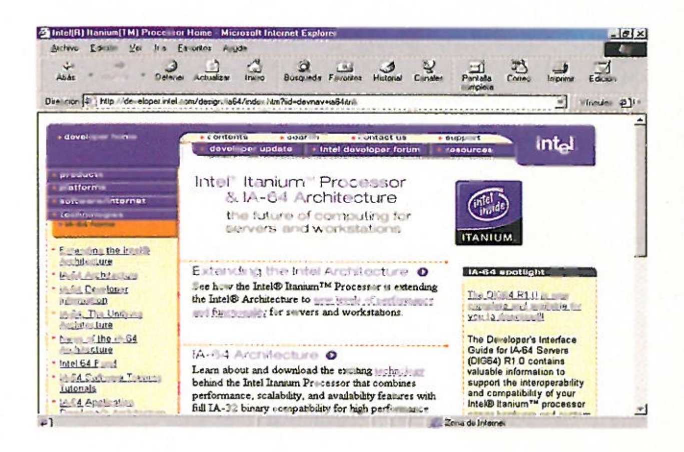
INTEL ENTREGA SU SISTEMA DE ITANIUM A LOS DESARROLLADORES

Intel Corp. ha entregado su nuevo sistema con prototipos de su nuevo procesador de *Itanium* a desarrolladores que puedan realizar nuevas aplicaciones que sean capaces de utilizar correctamente el nuevo procesador.

El *chip Itanium* estará disponible en el mercado a mediados del año 2000, siendo el primer procesador de esta compañía que usa arquitectura de 64 *bits*, y procesa datos en paquetes de 64 *bits* en lugar de los 32 estándar que se manejan actualmente. Los procesadores *Itanium* y toda su nueva familia de productos, estarán orientados hacia el mercado de grandes prestaciones y estaciones de trabajo con una gran carga a nivel computacional.

Aquellos desarrolladores que han tenido acceso a esta novedad, han podido comprobar la potencia de este nuevo *chip* desde hace algo más de un año gracias a un *software* que emula las prestaciones del *Itanium*, así como su arquitectura.

Par más información: **www.intel.com**



APARECE TIVOLI MANAGER PARA ORACLE 2.0

Tivoli presenta *Tivoli Manager* para *Oracle* 2.0, una solución orientada a reducir los gastos de operación al tiempo que asegura una alta disponibilidad y prestaciones para recursos críticos del negocio.

Se trata de una herramienta integrada por *Tivoli Enterprise* que aprovecha todas las ventajas de la arquitectura de tres niveles y gestiona la conectividad crítica entre los usuarios finales y las bases de datos, encuadrando cada paso en un proceso de transacción iniciado por el cliente. Esta aproximación única permite a los administradores de las bases de datos detectar problemas anticipadamente y resolver aquéllos surgidos entre los usuarios y las aplicaciones antes que afecten a la disponibilidad de la información y, en último término, al funcionamiento de operaciones críticas para el negocio.

Tivoli Manager para *Oracle* 2.0 cuenta con distintas capacidades como: un servidor centralizado de gestión de bases de datos, un seguimiento de éstas, así como su administración, o monitores preparados para paquetes de soluciones.

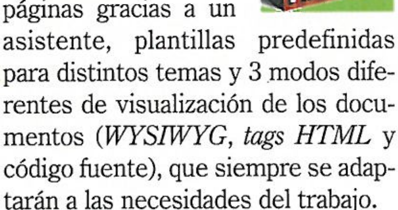
Para más información: **www.tivoli.com**

HERRAMIENTAS DE PROGRAMACIÓN

ENTORNOS DE PROGRAMACÓN

HOTMETAL PRO 6.0

Editor *HTML* que incorpora opciones para la publicación de las páginas *Web*. Facilita la creación de las páginas gracias a un asistente, plantillas predefinidas para distintos temas y 3 modos diferentes de visualización de los documentos (*WYSIWYG*, *tags HTML* y código fuente), que siempre se adaptarán a las necesidades del trabajo.



http://www.hotmetalpro.com/
Tfno: 91-637-18-51

HOTMETAL PRO 6.0

Editor de ficheros de texto que trabaja con ficheros *TXT*, *RTF*, *INI* y *REG* (Registro). Presenta una interfaz que soporta múltiples documentos (*MDI*), ya que puede mantener abiertos hasta 20 ficheros simultáneamente.

NEMO 1.01

Potente herramienta para el desarrollo de mundos *3D* interactivos. Con *NeMo* es posible añadir interactividad a entornos *3D*, importar modelos tridimensionales, animaciones, sonidos y texturas de los formatos más populares (*3D* Studio, *JPG*, *TIFF*, *TGA*, *BMP*, *PCX*, *DIB*, *AVI*, *ActiveMovie*, *WAV*, *MID*, *MP3*, etc.), probar y depurar las aplicaciones y divulgar su contenido.

SCRIBBLER 2000 V.1.9.0

Editor *JavaScript* y *VBScript* para generar *HTML* dinámico. Por su diseño y prestaciones puede ser utilizado tanto por programadores principiantes como avanzados. Presenta una interfaz multilingüe, una ventana de previsualización en la que contemplar el resultado del trabajo, una librería con 50 *scripts* predefinidos y funciones para su gestión.

TEXT PAD 4.1

Potente editor de texto para programadores. Corrige automáticamente los errores ortográficos más comunes utilizando comandos de "cambiar por", incluye una librería de fragmentos para insertar los textos usados con mayor frecuencia y guarda como macros las combinaciones de comandos más utilizadas.

WINLINUX 2000 FINAL BETA

Completa versión del sistema operativo *Linux*. Ha sido diseñada para ser instalada directamente en equipos que trabajan bajo *Windows*, y ejecutada como cualquier otra aplicación *Windows*. Incluye aplicaciones, herramientas de desarrollo, juegos, aplicaciones de *Internet*, etc.

LENGUAJES

VISUAL BASIC

ACTIVESKIN 3.0

ActiveX que permite cambiar completamente el aspecto, los diálogos y formularios de un proyecto. Ofrece una amplia variedad de herramientas para crear ventanas no rectangulares con formas dinámicas o animadas, añadir sombras, animaciones, *morphing* y efectos de semitransparencia.

BESTOFWARE SMARTUI 1.0.0005

Interfaz de usuario universal para *Visual Basic* 6. Es un *ActiveX* sin ventanas que da a las aplicaciones la apariencia y funcionamiento de *Windows* 2000, *Office* 2000, o cualquier otra interfaz de usuario que se desee.

CODE SMART 3.51

Nuevas funcionalidades para *Visual Basic* que proporcionan mejoras al *IDE*. Entre otras características el programa implementa: ítems de código coloreados, funciones de autoguardado, añade números de línea, realiza copias de seguridad de ficheros, una base de datos actualizable de código, etc.

VBEXPRESS LITE 1.6.0

Colección de 16 utilidades y una librería de código. Se integra con el *IDE* de *Visual Basic*. Ofrece gestión de errores, utilidades de codificación, funciones avanzadas de cortar y pegar, un verificador de formularios, un asistente de *tabs*, un verificador ortográfico y la posibilidad de generar código basado en plantillas personalizables.

JAVA

JAVASCRIPT EDITOR 2.6

Editor para la codificación en *JavaScript*. Presenta una interfaz dividida en cuatro áreas: en la parte superior izquierda, podemos visualizar una lista de todos los objetos *JavaScript* y pulsando sobre ellos aparecerán a la derecha sus propiedades y eventos.

JFORGE 2.62
Diseñador de interfaces gráficas para *Java*. Utiliza los componentes de *Java Foundation Class* y *Swing*. Cuenta con una interfaz *WYSIWYG* que facilita el trabajo e incluye funciones de edición de componentes, copiar, pegar y borrar, presenta una vista jerárquica de los componentes, etc.

KAWA 3.22
Entorno de desarrollo integrado para crear aplicaciones *Java*. Emplea un gestor visual de proyectos con un depurador gráfico, un editor de clases y un visor de código fuente. Soporta *JavaBeans*.

HTML

BASERUNNER 2.5.1
Programa *CGI* que permite a los usuarios de *Internet* acceder a bases de datos en formato *DBF*. Utiliza plantillas *HTML* y expresiones *xBase*. Soporta los ficheros estándar *dBase/FoxPro DBF* y permite acceder a los campos tipo memo. Ofrece soporte para *cookies*. Necesita un servidor *Web* que soporte *CGI*.

HTML COLORING GUIDE 2.0.137
Agiliza la creación de *BODY tags* de ficheros *HTML*. Permite seleccionar el color para el cuerpo del texto, establecer los parámetros *ALINK, LINK, VLINK*, y escoger una imagen de fondo. Luego, sólo hay que copiar el código generado al portapapeles y pegarlo en el documento *HTML*.

SPIDER WRITER 4.0
Editor *HTML* que incorpora numerosos asistentes y cajas de diálogo facilitan el diseño de los documentos, la introducción de *tags*, la creación de *JavaScripts* y el diseño de *frames* y tablas. Incluye opciones para realizar búsquedas y sustituciones de múltiples líneas, un verificador de enlaces, un cliente *FTP* interno, correctores de ortografía y sintaxis, y editores de mapas de imágenes, atributos y hojas de estilo.

OTROS

DYNAZIP COMPLETE 4.0
Colección de utilidades de compresión de archivos *ZIP*. De esta forma, los programas que desarrollemos pueden acceder a funciones de compresión y descompresión utilizando sencillas propiedades, métodos y eventos. Incluye componentes *ActiveX, VBX, OCX, DLL, VCL*, etc.

ELPACK 2.35
Colección de 18 componentes para programadores de *Delphi* 4. Con *ElPack* ya no será necesario incluir *COMTRLS.PAS* en los programas ni depender de las versiones de *COMCTL32.DLL*. Incluye una barra de avance de los procesos, un gestor de los ficheros utilizados recientemente, controles para dibujar gráficos, personalizar colores y gestionar esquemas de color.

ESB PROFESSIONAL COMPUTATION SUITE 1.1.0
Colección de 1.600 rutinas y 50 componentes para *Delphi* 4. Abarcan numerosas áreas como manipulación de fecha y hora, funciones trigonométricas, estadísticas, regresiones lineales, distribuciones de probabilidad, resolución de ecuaciones, fracciones, conversión de unidades, etc.

WINSOCK INTERFACE LIBRARY FOR C/C++ 3.0
Librería que simplifica el control de *Winsock* que proporciona acceso a *Internet*. Ofrece soporte para la mayoría de los protocolos habituales en *Internet* incluyendo *Finger, SMTP, POP3, FTP, NNTP* y *HTTP*. Incluye librerías, código fuente y aplicaciones de ejemplo. Necesita un compilador *C/C++*.

HERRAMIENTAS Y UTILIDADES

EXESCOPE 4.50
Editor de recursos para personalizar ficheros ejecutables. Permite modificar los diálogos, el diseño, las fuentes, el menú y el tamaño. Puede analizar y modificar *DLL, EXE, FON, OCX, VBX* y otros ficheros sin utilizar los ficheros fuente.

INTERSCOPE INSTALLMASTER 1.1 LITE
Herramienta de creación de programas de instalación. Comprueba automáticamente los requerimientos para distintos sistemas operativos, memoria, procesador, Registro, etc.

SECURITY ADMINISTRATOR 2.0
Utilidad de gestión de la seguridad de los equipos. Permite especificar distinto nivel de acceso para diferentes usuarios. Limita el acceso a elementos del panel de control como impresoras, redes, visualización y claves de acceso. También puede establecer *passwords* y bloquear por completo el ordenador.

SERVICE PACK MANAGER 4.2.1
Sistema de gestión de *Hotfixes* y *Service Packs* de *Windows NT*. Permite a los administradores de sistemas hacer un seguimiento y controlar *Hotfixes* y *Service Packs* en equipos individuales y redes. Muestra el estado de los equipos, imprime informes sobre el mismo y ofrece información detallada sobre las actualizaciones de *Microsoft* que se encuentran disponibles.

SIMAPP 1.03
Analiza y optimiza sistemas de control automático. *SimApp* es un programa adecuado para la simulación de sistemas de control basados en la idea de diagramas de bloques. Para los sistemas y subsistemas más utilizados es posible definir bloques

con sus propios símbolos y listas de parámetros, y guardarlos en barras de herramientas y librerías.

SIMCA-P 8.0

Programa para trabajar con modelos y análisis multivariable. Ofrece una amplia variedad de herramientas que facilitan el análisis de datos complejos y la presentación de los resultados generando unos gráficos sencillos de entender.



REDES

ADVANCED ADMINISTRATIVE TOOLS 3.1

Monitor de seguridad de redes que detecta, analiza y resuelve posibles problemas de seguridad en sistemas de *Internet*, *intranet* y *extranet*. Incorpora utilidades para identificar los servicios activos, monitorizar *CGI´s*, un verificador *proxy* y un verificador de listas de correo electrónico.

ESERV MAIL, NEWS, WEB, FTP AND PROXY SERVERS 2.80

Servidor *proxy* que incluye servidores *Web*, de noticias y correo electrónico. Facilita la conexión de una red *LAN* a *Internet* proporcionando servidores *proxy* para *FTP*, *HTTPS*, *SOCKS* y *DNS along with TCP and UDP port mapping*.

HSSPY 1.0

Herramienta de análisis de conexiones *TCP/IP*. Únicamente hay que introducir el nombre del dominio, el puerto *TCP/IP* y observar la transferencia de información. También puede analizar protocolos de *Internet* no basados en *Web* como *FTP*, *TELNET* y otros servicios.

MONITORIT 1.1

Herramienta de monitorización del rendimiento de sistemas y servidores *Windows*. Consta de dos módulos. El primero se instala en cada sistema cliente y recoge información del rendimiento del equipo en el que se encuentra. El segundo permite acceder a información histórica y en tiempo real de todas las operaciones y rendimiento del sistema cliente.

NORTON FHOST 6.0



Crea imágenes virtuales de la información del disco duro. La imagen obtenida puede ser copiada simultáneamente en varios ordenadores para crear instalaciones idénticas.

Permite elegir entre modo *GUI* o *BATCH*, comprimir las imágenes, establecer claves de acceso, recuperar ficheros, gestionar ficheros de imágenes, etc.

SNMP SWEEP 1.4

Escanea una amplia variedad de direcciones *IP*. *SNMP Sweep* además descarga la siguiente información de cada *IP*: nombre y descripción del sistema, tipo de equipo, localización, contacto, etc.

DOCUMENTACIÓN/ TUTORIALES

SÓLO PROGRAMADORES

AMZI! LOGIC EXPLORER 4.103

Intérprete y tutorial de *Prolog*. Se apoya en el proceso de creación de cuatro aplicaciones: un juego de aventuras, una base de datos inteligente, un sistema experto y un sistema de negocios. Incluye un depurador de código y ayuda en línea.

HARDWARE-LINX 2.01.1000

Recopilación de numerosas direcciones de fabricantes de *hardware* donde podemos encontrar sus respectivos *drivers*.

WEB RESOURCES TUTORIALS 6.3

Colección de tutoriales sobre programación de *JavaScripts*, *cookies HTML*... Además incluye una guía para crear *Active Channels*, formularios, una base de datos, botones de navegación, y un tutorial para publicar nuestra página *Web* en motores de búsqueda.

ATENCIÓN:
En caso de problemas con el CD-ROM envíelo por correo ordinario, a la atención del SERVICIO TÉCNICO DE SÓLO P., incluyendo en el interior del sobre sus datos personales, a la siguiente dirección:
C/ San Sotero, Nº 5, 1ª Planta,
28037 (Madrid)

IMPRESCINDIBLES

ANTIVIRUS

- AntiViral Toolkit Pro 3.0.129
- AVP Virus Enciclopedia
- McAfee VirusScan 4.0.3
- Panda Antivirus 6.10.00 Platinum

GRÁFICOS

- Icon Bank 4.0 Gold Edition Web Edition / Desktop Edition
- IconForge 4.6
- MicroAngelo 98 v4.77
- Paint Shop Pro with Animation Shop 6.0
- Reptile 2.0
- SureThing CD Labeler 2.0
- ThumbsPlus 4.02
- Ultra Fractal 2.04
- Xara WebStyle 1.2

INTERNET

- Añadir Pro 4.01.001
- AutoWinNet 6.0 Beta 1
- Copernic 2000 v4.0
- Cuentapasos 3.75
- CuteFTP 3.5
- Dial-Up Magic 1.8
- Eudora Light 3.0.6
- GetRight 4.1.1
- Go!Zilla 3.5
- Guardián 1.1
- HomeSite 4.01
- ICQ 99b beta 3.19 build 2569
- MIRC 32 5.61
- URL Organizer 2.3.1
- WebZIP 3.06
- WinGate 3.0.5

MULTIMEDIA

- AudioCatalyst 2.01
- CDH Media Wizard 4.12
- COWON Jet-Audio 4.6
- Sonique 1.30
- WinAMP 2.50e

NAVEGADORES

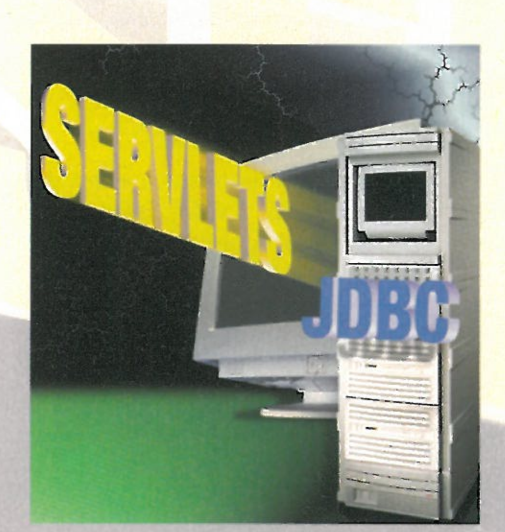
- Internet Explorer 5.01
- NeoPlanet 5.0.0.1045
- Netscape Communicator 4.7
- Opera 3.60

PROGRAMACIÓN

- Decafe Pro 3.6
- Free Pascal v0.99.12
- Hackman 4.02
- Help & Manual 2.25
- InstallConstruct 3.2.1
- Java Development Kit 1.2.2
- UltraEdit Professional Text/HEX Editor 6.20b
- Windows Registry Guide 1.3
- WinHex 8.85

UTILIDADES

- Adobe AcrobatReader 4.0
- Advanced Registry Tracer (RegFix) 1.11
- Babylon Translator 2.1
- CallCenter 3.5.8
- Calendar Builder 3.2j
- DirectX 7.0
- Emergency Recovery System 8.73
- Nero 4.0.7.5
- SiSoft Sandra 99.8.5.30
- Where Is It? 2.15a
- Windows Commander 4.01
- WinZip 7.0



Java Servlets y JDBC (I)

Adolfo Aladro García.
Analista Programador

Java ha irrumpido con fuerza en el mundo de la programación y uno de los entornos donde está cobrando mayor importancia es en el de las aplicaciones *Web*. Los *servlets,* junto con el acceso a bases de datos mediante *JDBC,* son una de las opciones más prometedoras para desarrollar sitios *Web* complejos, potentes y versátiles.

LOS JAVA SERVLETS

Hasta el momento la parte más conocida de *Java* son los *applets*, pequeños programas que se cargan a través de *Internet* y se ejecutan en la máquina cliente, con independencia de la plataforma.

En una primera aproximación podríamos decir que los *servlets* son al servidor lo que los *applets* al navegador del cliente. La diferencia principal con respecto a estos últimos es que son objetos sin rostro, es decir, no tienen interfaz gráfica.

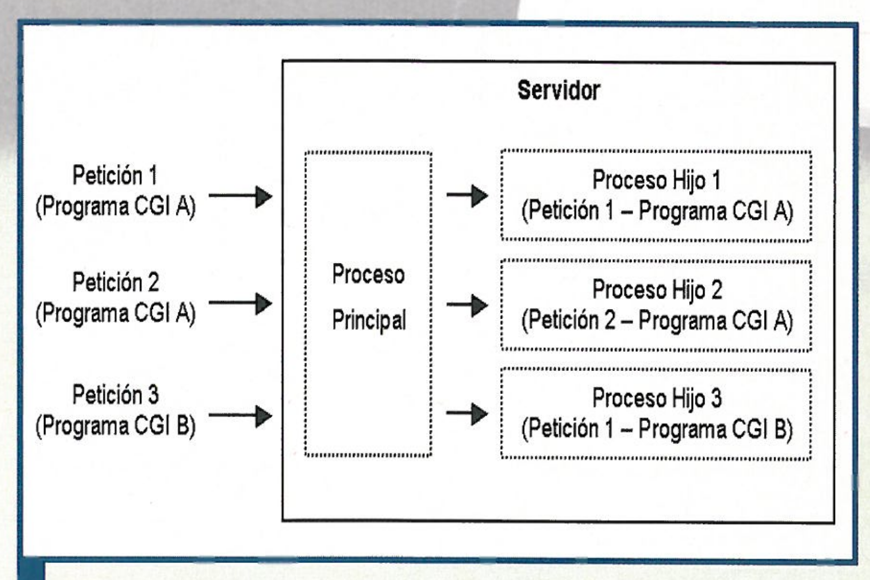


Figura 1.- Esquema de funcionamiento de una aplicación *Web* desarrollada con programas *CGI*.



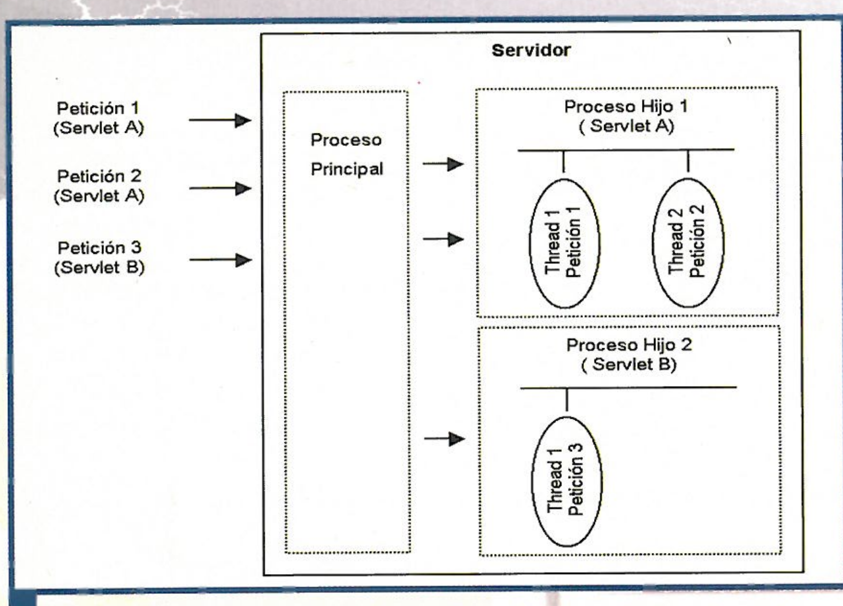


Figura 2.- Esquema de funcionamiento de una aplicación *Web* desarrollada con *Java servlets.*

Se cargan dinámicamente en el lado del servidor extendiendo las funcionalidades del mismo y son independientes de la plataforma. Los *servlets* pueden realizar una gran variedad de funciones entre las que pueden citarse:

- Crear y devolver páginas *HTML* cuyo contenido puede generarse dinámicamente en función de la petición del cliente (lo que se denominan *páginas dinámicas*. Tradicionalmente se han venido realizando mediante programas basados en la interfaz *CGI*).
- Comunicarse con otros recursos situados en la máquina servidor, por ejemplo bases de datos, con otras aplicaciones *Java* y en general con cualquier otro programa escrito en cualquier lenguaje, lo que incluye el acceso mediante *JNI* (*Java Native Interface*) a librerías desarrolladas en *C*.
- Mantener conexiones con múltiples clientes, aceptando datos de entrada procedentes de todos ellos mediante *broadcasting*.
- Establecer una conexión desde el servidor con un *applet* en el navegador del cliente, mantenerla abierta y permitir la transmisión de varios datos a través de esa única conexión. El *applet* también puede iniciar la conexión entre el navegador y el servidor, lo que incrementa notablemente la eficiencia de la comunicación cliente-servidor, y además es independiente del protocolo: puede utilizarse un protocolo personalizado o uno estándar.
- Filtrar los tipos *MIME* para llevar a cabo una cierta operación, como por ejemplo la conversión de imágenes.

LIMITACIONES DE LOS PROGRAMAS CGI

Pese a que los programas *CGI* suponen una forma sencilla de poder dotar de dinamismo a las páginas *Web*, en general no representan un buen sistema. Esto se debe a que cada vez que un cliente realiza una petición que implica a un programa *CGI*, el servidor lanza un nuevo proceso, con el consumo de recursos que esto implica (memoria, *CPU*, etc).

Los servlets pueden comunicarse con otros servlets o con applets

Otra desventaja de la programación *CGI* surge de una de las características del protocolo *HTTP*. Se

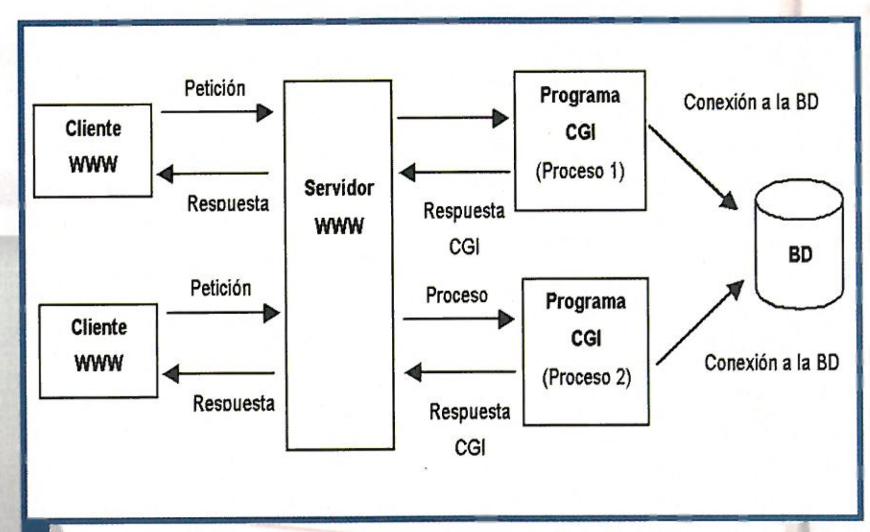


Figura 3.- Modelo de aplicación *Web* con acceso a bases de datos basado en programas *CGI*.

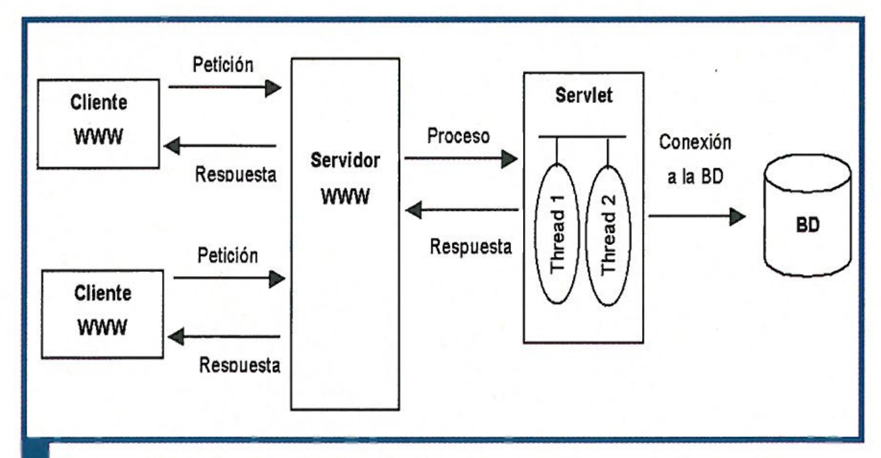


Figura 4.- Modelo de aplicación *Web* con acceso a bases de datos basado en *Java servlets.*

trata de un protocolo sin estado, de tal manera que cada conexión que se realiza desde un cliente con un servidor es siempre como si fuera la primera y no deja un estado que permita diferenciarla de las que se realizarán a continuación. Hay que recordar que la actualización de la ventana del navegador con el resultado de ejecución del programa *CGI* implica la terminación del mismo y el cierre de la conexión *HTTP*.

DIFERENCIAS ENTRE LOS PROGRAMAS CGI Y LOS SERVLETS

Los *servlets* son persistentes, independientes de la plataforma e incorporan todo tipo de características avanzadas tales como seguridad, acceso a bases de datos e integración con *applets Java*. Por todo ello, la programación basada en *servlets* presenta ventajas considerables con respecto a los programas *CGI* tradicionales desarrollados en lenguajes tales como *Perl* o *C*.

Independencia de la plataforma

Los *servlets* pueden correr en cualquier plataforma sin tener por qué volver a ser reescritos. Por el contrario, la mayoría de los programas *CGI* no son portables. Están realizados para una plataforma concreta y las posibilidades de reutilización son casi siempre costosas.

Rendimiento

Respecto al rendimiento los programas *CGI* presentan serias deficiencias. Cada vez que un servidor recibe una petición *CGI* necesita lanzar un proceso diferente, esperar a que termine de ejecutarse, cerrarlo y entonces, enviar el resultado al navegador. Si estamos accediendo a una base de datos, por ejemplo, esto implica abrir y cerrar una nueva conexión a la base datos cada vez que el programa *CGI* se ejecuta.

Por el contrario, los *servlets* sólo necesitan cargarse una vez. No crean un nuevo proceso cada vez que son invocados, sino que crean un hilo de ejecución de proceso (un *thread*) por cada una de las peticiones y cada hilo se ejecuta concurrentemente. Siguiendo con el ejemplo de acceso a bases de datos, un *servlet* sólo crea una conexión a la base de datos y a través de ésta es capaz de atender a las distintas peticiones que le van llegando.

Un servlet crea hilos de ejecución por cada petición

Hay una única *Máquina Virtual Java* corriendo en el servidor y el *servlet* se carga una sola vez cuando es invocado. No se vuelve a cargar de nuevo a no ser que cambie -se retoque el código fuente y vuelva a ser compilado-. El *servlet* queda residente en memoria por lo que funciona a gran velocidad. Esto también implica que diferentes llamadas al *servlet* pueden compartir información, lo que facilita además el mantenimiento de un área común de datos entre dis-

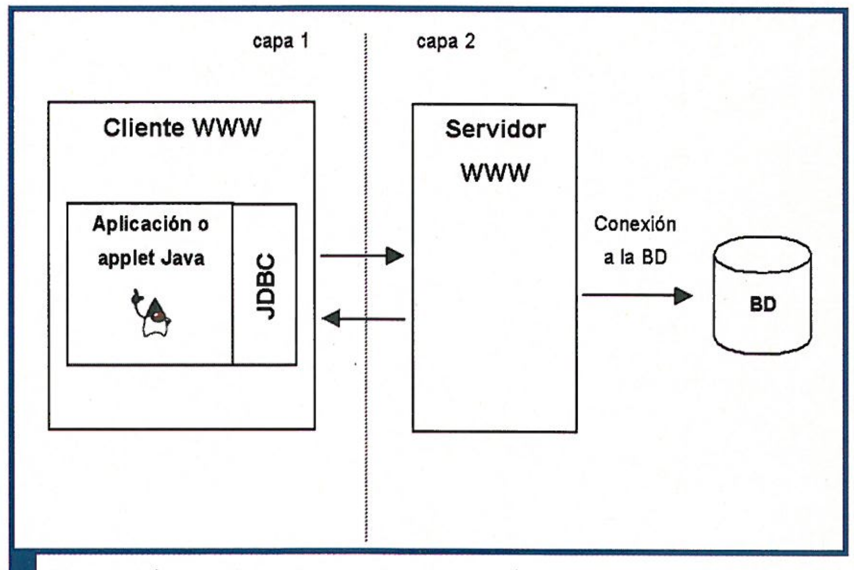


Figura 5.- Aplicación *Web* en dos capas con *JDBC* y *servlets.*

tintos usuarios que invocan al mismo *servlet*.

- **Comunicación**

Los *servlets* pueden comunicarse entre sí y al mismo tiempo con *applets* que se estén ejecutando en la máquina cliente. De esta forma la carga de un proceso costoso puede distribuirse entre el cliente y distintas máquinas servidor.

- **Unificación, estandarización y mantenimiento**

Los *servlets* utilizan una *API* estándar de *Java* que proporciona unas clases con las que ponerse a trabajar inmediatamente. De esta manera podemos estructurar y unificar todos los programas *CGI*, sea cual sea su naturaleza: acceso a bases de datos, recepción de formularios y su posterior envío a través del correo electrónico, etc.

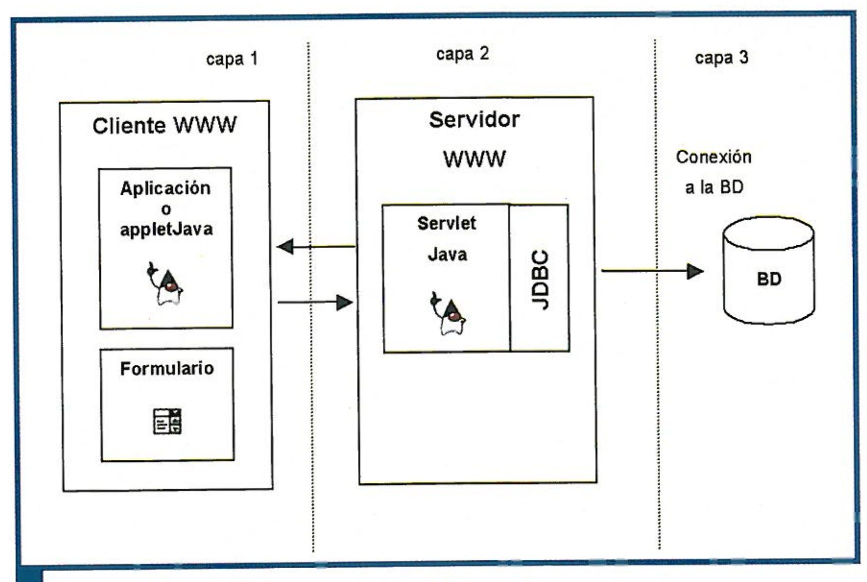


Figura 6.- Aplicación *Web* en dos capas con *JDBC* y *servlets*.

Así se evita la engorrosa situación actual en la que se tienen algunos programas hechos en *Perl*, otros en *C*, algunos más en algún lenguaje *script* de la máquina servidor, etc, circunstancia que encarece notablemente todos los procesos de mantenimiento, actualización o migración entre plataformas.

En definitiva, *Java* proporciona un soporte mejor para el desarrollo de aplicaciones *Web*, lo que en consecuencia conduce a reducir el

número de errores en comparación al desarrollo con lenguajes como *C*, *Perl* o cualquier otro lenguaje de programación. Utilizando la *API* estándar para *servlets* el programador no ha de preocuparse del funcionamiento interno del servidor. Los datos procedentes de un formulario, las cabeceras del servidor, las *cookies*, etc, son manejados por las clases *Java* que existen por debajo del *servlet*.

APLICACIONES WEB CON SERVLETS Y JDBC

Desde el punto de vista de la arquitectura cliente/servidor, existen fundamentalmente dos formas de abordar el desarrollo de aplicaciones *Web* que accedan a bases de datos mediante la utilización de *servlets* y *JDBC*: el modelo en dos capas (*two-tier model*) y el modelo en tres capas (*three-tier model*). Las dos soluciones son muy distintas y conllevan implicaciones que hacer cambiar notablemente la estructura general de la aplicación *Web* que estemos desarrollando.

MODELO EN DOS CAPAS (TWO-TIER MODEL)

En una arquitectura cliente/servidor clásica tenemos dos "capas" (*two-tier*): una donde está el cliente, que implementa la interfaz, entre otras cosas; y, otra donde se encuentra el gestor de bases de datos que trata las peticiones recibidas desde el cliente.

La lógica de la aplicación se encuentra por lo tanto repartida entre el cliente y el servidor. Un ejemplo de esta configuración podría ser el siguiente: un *applet Java* se carga en el navegador del cliente y trabaja directamente con la base de datos mediante *JDBC*.

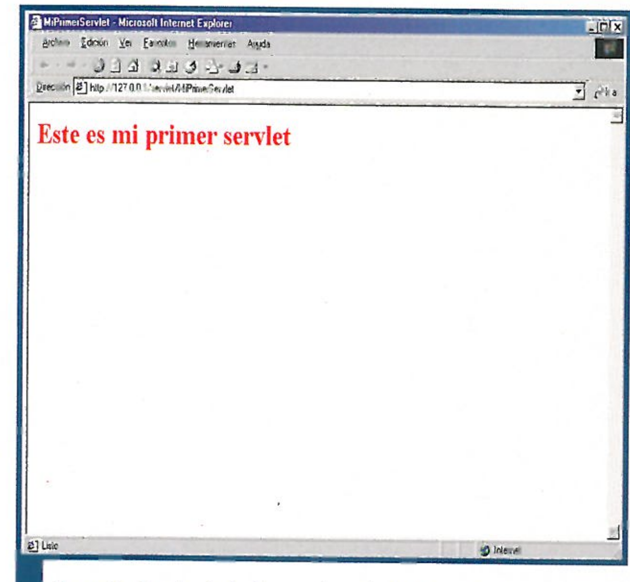


Figura 7.- Resultado de llamar al *servlet MiPrimerServlet*.

Ventajas

- Se mantiene una conexión persistente con la base de datos.
- Se minimizan las peticiones en el servidor trasladándose la mayor parte del trabajo al cliente.
- Se gana en rendimiento gracias a la conexión directa y permanente con la base de datos. A través de una única conexión se realiza el envío y recepción de varios datos.

Inconvenientes

- La primera desventaja, y probablemente la más importante, es que esta solución es muy dependiente del tipo controlador *JDBC* que se utilice para acceder a la base de datos. El acceso se realiza desde el cliente y esto signifi-

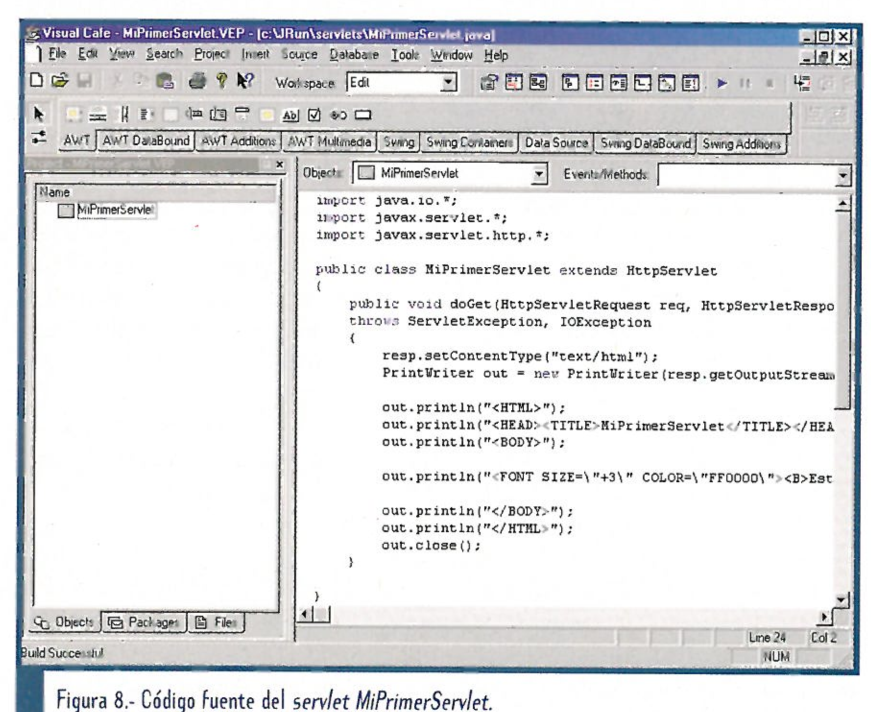


Figura 8.- Código fuente del *servlet MiPrimerServlet*.



ca que es él el que tiene que tener instalado en su sistema los controladores necesarios para que se produzca la comunicación con la base de datos.

- Además hay que tener en cuenta que el modelo de seguridad de *Java* impide que desde un *applet* sin validar –lo que se conoce como *untrusted applet*-, como lo son la mayoría de los que se ejecutan en un navegador, se puedan realizar las siguientes operaciones:

1. El acceso general, y por supuesto mediante *JDBC*, a bases de datos situadas en direcciones *URL* distintas a aquellas de las que procede el mismo *applet*.

2. La configuración de recursos locales como, por ejemplo, la información de la fuente de datos *ODBC* para usar el puente *JDBC-ODBC*.

3. La descarga de clases nativas: o sea, aquellas cuyo nombre empieza por **java**. Esta restricción afecta directamente a los navegadores que utilizan *JDK 1.0.2* o anterior, pues *JDBC* es posterior a esta versión, de forma que las clases apropiadas no estarán instaladas localmente ni podrán ser descargadas de *Internet* por el *applet*.

- Finalmente debemos tener en cuenta que es bien conocido que los programas *Java* pueden ser descompilados muy fácilmente con lo que introducir el acceso a nuestras bases de datos mediante un *applet Java* conlleva un riesgo considerable en cuanto a la seguridad.

Por poner un ejemplo, baste mencionar que si queremos acceder desde un *applet Java* a una base de datos, es evidente que en el código fuente del *applet* aparecerán el usuario y la contraseña con la que nos conectamos a la base de datos. Esta información, obviamente, no es deseable que sea de conocimiento público.

A la hora de trabajabar, la arquitectura en dos capas resulta insegura

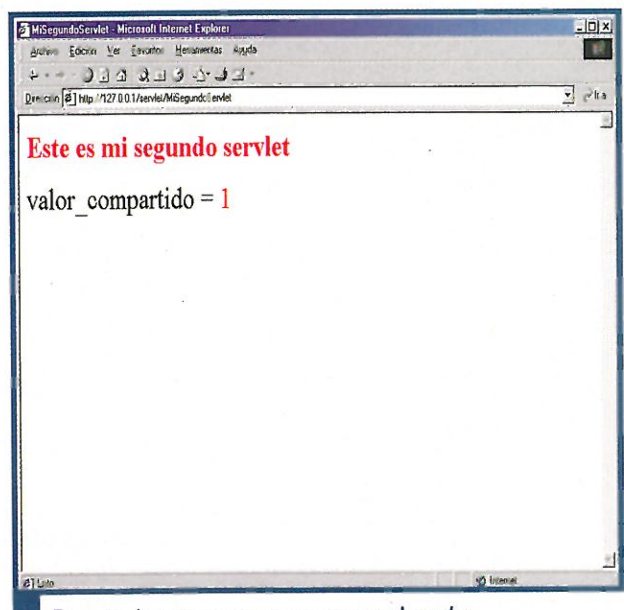


Figura 9.- La primera vez que se ejecuta el *servlet MiSegundoServlet* muestra el valor inicial de la variable compartida.

De todo lo anterior se deduce que en general el acceso a bases de datos mediante *applets Java* será una solución idónea preferiblemente para *intranets* o sistemas cerrados, donde se tiene conocimiento de quiénes son los que van a acceder a las bases de datos.

MODELO EN TRES CAPAS (THREE-TIER MODEL)

Con la arquitectura cliente/servidor en tres capas (*three-tier*) añadimos una nueva capa entre el cliente y el servidor donde se implementa la lógica de la aplicación. De esta forma el cliente es básicamente una interfaz, que no tiene por qué cambiar si cambian las especificaciones de la base de datos o de la aplicación. Queda aislado completamente del acceso a los datos.

Así, un *applet Java* se carga en el navegador del cliente y se comunica con un *servlet* que corre en la máquina servidor; o bien accedemos a la base datos a través de un formulario *HTML*. El *servlet* establece una conexión la base datos mediante *JDBC*.

Ventajas

- No existe ningún problema con respecto al tipo de controlador *JDBC* utilizado para acceder a la base de datos. Todos los recursos necesarios para establecer la conexión con la base de datos se encuentran en el servidor y por lo tanto, el cliente no necesita instalar nada adicional en su máquina para poder acceder a la base de datos.

- En definitiva, esta arquitectura proporciona considerables mejoras desde el punto de vista de la portabilidad de la aplicación, escalabilidad, robustez y reutilización del código. Asimismo facilita las tareas de migración o cambios en el sistema gestor de la base de datos.

- Desaparecen las restricciones derivadas de las limitaciones de los *applets* impuestas por el modelo de seguridad de *Java*.

Inconvenientes

- Esta solución es algo menos eficiente que la propuesta en el modelo anterior ya que hemos añadido una capa intermedia más de *software*.

INTRODUCCIÓN A LA PROGRAMACIÓN DE SERVLETS

A lo largo de los artículos de esta serie profundizaremos en todos los aspectos de la programación de *servlets* así como en la utilización de *JDBC* para acceder a bases de datos. Lo que resta de este primer artículo lo dedicaremos a familiarizarnos con los *servlets* a través de pequeños ejemplos muy sencillos. Posteriormente, a medida que nos adentremos en la *API*, se entenderán mucho mejor todos aquellos aspectos que ahora puedan quedar un poco confusos.

PREPARANDO EL ENTORNO

Antes de pasar a realizar ningún ejemplo, lo primero que debemos hacer es preparar el entorno de trabajo. Para emular una arquitectura cliente/servidor en nuestro *PC* necesitamos un servidor. Además, este servidor tiene que ser capaz de ejecutar *servlets*. Finalmente, será preciso contar con alguna herramienta de desarrollo *Java* para escribir y compilar los *servlets*.

En una plataforma como *Windows 95/98* o *Windows NT* podemos utiliza el servidor que distribuye gratuitamente *Microsoft*: *PWS* (*Personal Web Server*). Si bien este servidor no puede ser utilizado en sistemas reales dadas sus limitadas capacidades, es más que suficiente para aquellos casos en los que su utilidad no es más que didáctica.

El esqueleto de los servlets es siempre el mismo

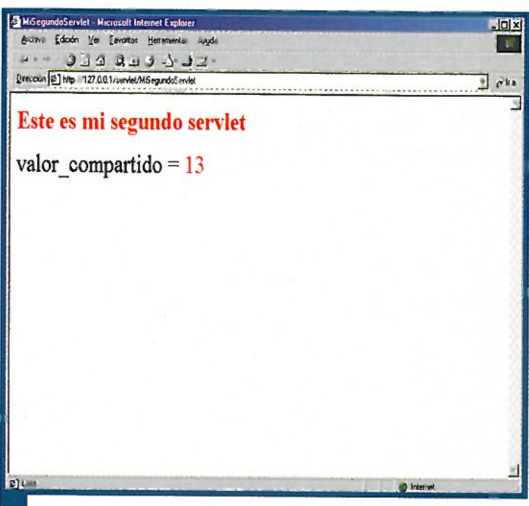


Figura 10.- La variable compartida se incrementa por cada llamada *servlet*, ya que éste se encuentra en memoria.

En el caso de trabajar con *Windows NT* encontraremos el servidor en el *Option Pack*, que también puede descargarse gratuitamente del sitio *web* de *Microsoft*. El *CD-ROM* de *Windows 98*, en el directorio **\add-ons\pws** se halla el archivo **setup.exe**. Ejecutándolo comienza la instalación del servidor.

Los programas CGI consumen muchos recursos

La dirección *IP* de nuestro ordenador es la **127.0.0.1** y podremos acceder a las páginas *Web* a través de la dirección *URL* *http://127.0.0.1/* o bien utilizando el nombre del equipo, por ejemplo, *http://dit6a6/*.

Necesitamos añadir un plug-in al servidor para ejecutar servlets

El servidor *PWS* no está preparado para ejecutar *servlets* por lo que es necesario instalar en nuestro ordenador un *plug-in*. En este caso hemos optado por *JRun* (**www.allaire.com**). Este *plug-in* se distribuye gratuitamente y su potencia y versatilidad hacen de él uno de los más utilizados incluso en entornos profesionales. La instalación de *JRun* es muy sencilla siguiendo las instrucciones.

En el directorio *C:\JRun\servlets* se encuentran los *servlets*. Una vez que todo está instalado, nuestro servidor está activo y los servicios de *JRun* están activados, podemos pasar a ejecutar nuestro primer *servlet* de ejemplo.

NUESTRO PRIMER SERVLET

Si tecleamos en el navegador la dirección:

```
http://127.0.0.1/servlet/MiPrimer-
    Servlet
```

podemos observar el resultado de la ejecución. Es el clásico programa "Hola Mundo" con el que todos los programadores empiezan a aprender un nuevo lenguaje.

En primer lugar deben aparecer las sentencias *import* que indican las clases que van a ser utilizadas. Las dos últimas son las específicas para los *servlets*.



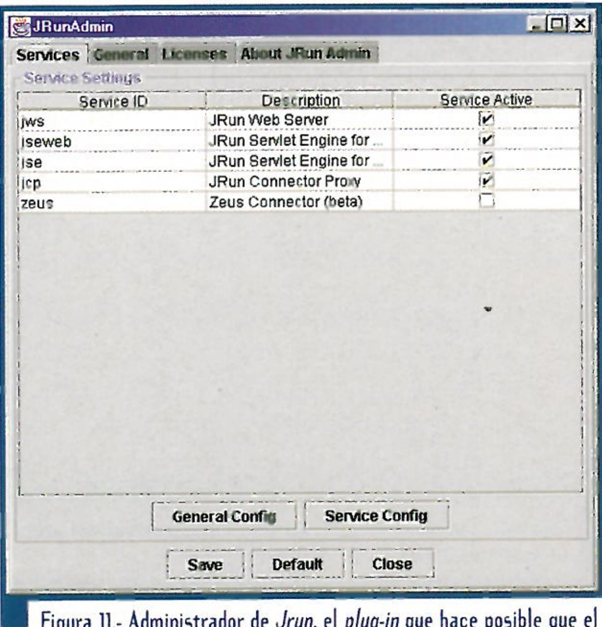


Figura 11.- Administrador de *Jrun*, el *plug-in* que hace posible que el servidor *Web* pueda ejecutar *servlets*.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Los *servlets* que se utilizan en arquitecturas cliente/servidor basadas en el protocolo *HTTP* deben extender la clase *HttpServlet*, y así se indica a continuación en el código fuente:

```
public class MiPrimerServlet
    extends HttpServlet {
...
}
```

El procedimiento *doGet* es el principal y se encarga de procesar las peticiones que llegan al *servlet*.

```
public void doGet(HttpServletRe-
    quest req,
    HttpServletResponse resp)
        throws ServletException,
    IOException {
...
}
```

Existen otros métodos como *Service* o *doPost*, que se utilizan en aquellos casos en los que se espera que los datos lleguen al *servlet* mediante un método *POST*, pero por el momento basta con saber que todo *servlet* debe contar con un procedimiento principal que es el que trata las peticiones recibidas.

La utilización de JDBC en los servlets puede considerarse como segura

Los parámetros *req* y *resp* del método *doGet* hacen referencia a los objetos de tipo *HttpServletRequest* y *HttpServletResponse* respectivamente. Estos objetos sintetizan a través de sus propiedades y métodos todas las necesidades involucradas en la tarea de recibir peticiones y responder a éstas.

El modelo en dos capas depende de los controladores JDBC

La primera línea del procedimiento *doGet* tiene como propósito indicar al cliente el tipo de la respuesta ofrecida. Para ello se utiliza el método *setContentType* del objeto *resp*.

```
resp.setContentType("text/html");
```

El valor **"text/html"** indica que la respuesta del *servlet* será con formato de código *HTML*.

Antes de generar la respuesta del *servlet* es preciso establecer un canal de escritura. Los canales de escritura y/o lectura son el medio natural dentro de *Java* para llevar a cabo procesos de entrada y salida de datos.

```
PrintWriter out =
    new PrintWriter(resp.getOut-
    putStream());
```

El método *getOutputStream* del objeto devuelve un canal mediante el que podemos escribir los datos que finalmente aparecerán en el navegador del cliente. Este canal puede "reconvertirse" a uno de tipo *PrintWriter* simplemente utilizando el constructor de esa clase.

```
out.println("<HTML>");
...
out.println("</HTML>");
```

Las siguientes líneas son una serie de llamadas al método *println* mediante las que se va generando la página *HTML* correspondiente.

El método setContentType se encargará de determinar el tipo MIME devuelto

Finalmente, la última sentencia se encarga de cerrar el canal de lectura abierto.

```
out.close();
```

Este *servlet* no sirve para gran cosa pero representa el esqueleto básico de casi todos. Lo más importante es entender la manera en la que este *servlet* es ejecutado por el servidor cada vez que recibe una petición.

El servidor carga en memoria el *servlet* la primera vez que es invocado. Entonces inicializa todos los valores que sean necesa-

rios y ejecuta el procedimiento *init*, si es que existe (ya hemos visto en el ejemplo anterior que nuestro *servlet* carecía de él). Una vez que todo el proceso ha concluido pasa a ejecutar el procedimiento *doGet*.

Java permite un mejor mantenimiento de las aplicaciones Web

Posteriormente, cada vez que el servidor recibe una petición referida a ese *servlet*, se abre un nuevo hilo de proceso donde se ejecuta exclusivamente el procedimiento *doGet*. Es decir, los valores de la clase no vuelven a inicializarse otra vez, ni el procedimiento *init* vuelve a ejecutarse. Este modo de funcionamiento puede observarse fácilmente siguiendo nuestro segundo ejemplo, *MiSegundoServlet*.

Hemos modificado el código fuente de manera que ahora la clase tiene un miembro llamado *valor _compartido*.

```
public class MiSegundoServlet
  extends HttpServlet {
    int valor_compartido = 1;
  ...
```

Éste es un número entero y cada vez que el *servlet* es invocado se muestra en la página y se incrementa en una unidad su valor.

```
out.println("<FONT SIZE=\"+3\"
     COLOR=\"FF0000\">" +
   valor_compartido + "</FONT>");
valor_compartido++;
```

El resultado es que la primera vez que el *servlet MiSegundoServlet* es invocado el valor de *valor_compartido* es igual a **1**. Pero las subsiguientes llamadas muestran el resultado de incrementar ese valor, tal y como puede apreciarse en la Figura 10. Si ahora compilásemos de nuevo el *servlet* el valor volvería a inicializarse ya que de nuevo se trataría de la primera petición, y por lo tanto se ejecutaría todo su código.

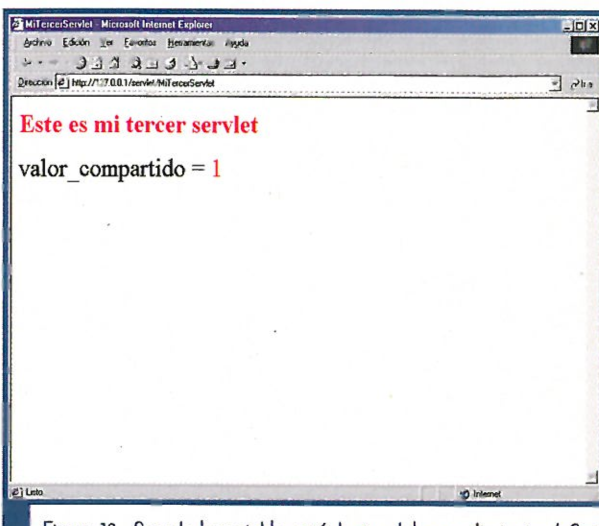


Figura 12.- Cuando la variable está dentro del procedimiento *doGet* ya no es compartida por las distintas peticiones.

El procedimiento doGet se ejecuta por cada petición

Comprender este comportamiento es muy importante ya que cuando pasemos a insertar dentro de los *servlets* el código que accede mediante *JDBC* a las bases de datos será preciso estudiar en qué lugares escribiremos los bloques de código que realizan la conexión a la base de datos, las peticiones a la misma y la desconexión.

Los servlets permiten compartir información entre peticiones

Además, el hecho de tener zonas de almacenamiento que pueden ser compartidas por distintas peticiones de manera simultánea, nos hace plantearnos el tema de la seguridad. ¿Qué ocurre si dos *servlets* quieren escribir al mismo tiempo la variable *valor_compartido*? En los próximos artículos nos adentraremos en todos estos detalles.

Echemos ahora un vistazo al tercer ejemplo, *MiTercerServlet*. La única diferencia con respecto al primero consiste en el hecho de que la declaración de la variable *valor_compartido* se encuentra ahora dentro del procedimiento *doGet*.

```
public void doGet(HttpServletRe-
    quest req,
   HttpServletResponse resp)
      throws ServletException,
    IOException {
        int valor_compartido =
    1;
  ...
```

Ahora, por muchas veces que se ejecute el *Servlet* el resultado será el mismo. El valor de *valor_compartido* es siempre **1**. El procedimiento *doGet* se ejecuta cada vez que llega una petición al servidor en un hilo de ejecución que no tiene nada que ver con los demás. Por lo tanto siempre se inicializa la variable *valor_compartido*. En realidad esta variable no es compartida por las distintas peticiones, tal y como ocurría en el segundo de nuestros *servlets*.



SUSCRÍBETE AHORA Y OBTENDRÁS...

1 • El **pack completo COREL DRAW 8.0**. COREL DRAW, COREL DRAW PHOTO-PAINT 8, COREL OCR-TRACE 8, COREL DREAM 3D 8, COREL TUTOR, COREL CAPTURE 8, COREL TEXTURE 8, COREL SCAN 8, COREL SCRIPT EDITOR, BITSTREAM FONT NAVIGATOR, más de 40.000 imágenes y símbolos, 1.000 fotografías, 1.000 fuentes True Type y Tipo 1, 750 objetos y 450 plantillas de COREL DRAW.

2 • **50 CD-ROM** con más de 200 programas y utilidades de diseño gráfico, 500 fotografías profesionales y 49 Webs de diseño para navegación, necesarias para el seguimiento del máster junto con su código de ejercicios.

3 • **50 fascículos** coleccionables divididos en dos partes diferenciadas: Teoría y Práctica, que componen más de 1.000 páginas del más completo manual de diseño gráfico con COREL DRAW 8.0, abarcando el uso del color y tipografías, filtros, técnicas de compresión y diseño Web.

4 • **TUTOR ON LINE**, que resolverá todas las dudas que pueden ir surgiendo durante el máster.

5 • **DIPLOMA** acreditativo, de que se ha recibido y aprovechado el Máster COREL DRAW de Diseño Gráfico.

6 • **6 CARPETAS** para almacenamiento de fascículos y CD-ROM.

Con él aprenderás a:

- Escanear fotografías y realizar retoques fotográficos.
- Combinar todos los elementos gráficos en una publicación.
- Crear gráficos optimizados para Internet, mapas de imágenes, etc.
- Exportar imágenes para su uso en páginas Web.
- Utilizar todo tipo de software gráfico especializado.
- Usar correctamente las tipografías en folletos, posters, etc.
- Diseñar animaciones y realizar el rodaje final.
- Comprender los diferentes sistemas de postproducción e impresión.
- Crear y componer escenas en 3D.
- Emplear todos los dispositivos de hardware relacionados con el diseño gráfico.

... y mucho más.

BOLETÍN DE SUSCRIPCIÓN OFERTA VÁLIDA SÓLO PARA ESPAÑA

SI, deseo suscribirme al **MASTER COREL DRAW**

CONDICIONES DE PAGO: (MARQUE LA OPCIÓN DESEADA)

❑ Pago al contado: A partir de la tercera entrega.
29.990 Ptas.
AHORRO con respecto a precio en quiosko: más de 14.000 ptas.
Se entregará una versión completa de Corel DRAW 8.0 en el primer envío
❑ Pago a plazos:
Primer envío: 16 primeros fascículos por **9.750** ptas.
Resto: 8 mensualidades de **3.250** ptas.
Corel DRAW 8.0 se incluirá en los envíos 7 y 12
Precio en el Quiosko: 795 Ptas. cada entrega de 1 fascículo y un CD-ROM
Total 50 fascículos, 50 CD-ROM y 6 tapas: 44.125 ptas.

UTILICE MAYÚSCULAS PARA RELLENAR ESTA TARJETA

Nombre y apellidos ....F. nacimiento.......................

Domicilio .... C.P. ....

Ciudad .... Provincia ....Telf. ....

e-mail .... Profesión ....

FORMAS DE PAGO:

❑ Con **cargo a mi tarjeta VISA** Nº ____ ____ ____ ____ Caduca ....

❑ **Domiciliación bancaria**

Sr. Dtor. del banco ....

Población ....

CÓDIGO CUENTA CORRIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Ruego a UD. que se sirva cargar en mi ❑ cuenta corriente ❑ libreta de ahorro el/los recibos que le serán presentados por REVISTAS PROFESIONALES, S.L. como pago de mi suscripción al **MASTER COREL DRAW DE DISEÑO GRÁFICO**

❑ **Cheque** a nombre de REVISTAS PROFESIONALES, SL.

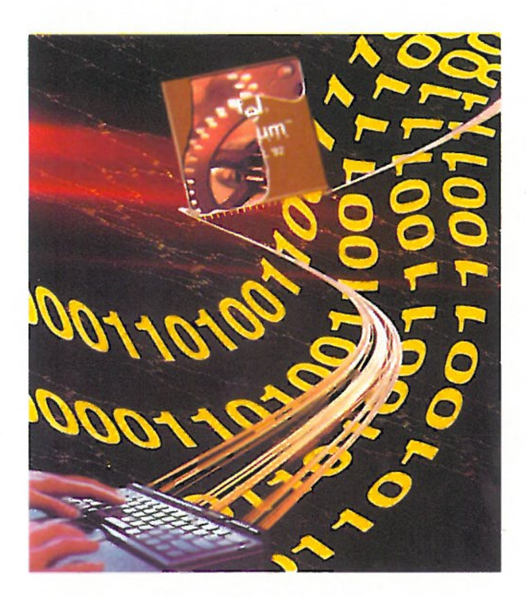
❑ **Contra reembolso** del importe más gastos de envío.

❑ **Giro Postal** (adjunto fotocopia del resguardo)*.

Firma

*Esta forma de pago es exclusivamente para pagos al contado

Programación de Threads (II)



Juan Luis Ceada Ramos.
Programador en ARCABE Formación y Servicios Informáticos.
Octavio Martín Díaz.
Profesor de Análisis de la Universidad de Sevilla.

En esta entrega veremos cómo compartir recursos desde varios *threads* mediante los diferentes mecanismos que *Delphi* y *Windows* nos proporcionan.

CONSIDERACIONES INICIALES

En principio, la regla máxima consiste en proteger el acceso a los recursos compartidos. Es decir, siempre habrá que tener especial cuidado con los recursos utilizados por varios *threads*, como por ejemplo el uso compartido de una impresora. Si dos *threads* quisiesen imprimir a la vez en la misma impresora, tendríamos muchos problemas.

El primer **thread** que consiguiese el control de la impresora debería bloquear el acceso al otro *thread,* por lo menos hasta que el primero termine de enviar los datos a la impresora.

Para evitar errores provocados por el acceso a recursos compartidos sin control, tendremos que bloquear la ejecución del resto de los *threads* que acceden a dichos recursos hasta que el que

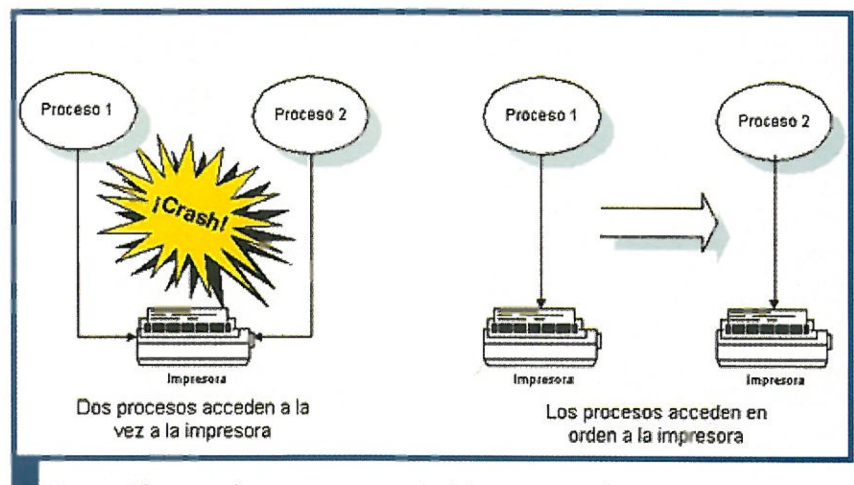


Figura 1.- El acceso a los recursos compartidos debe ser secuencial.



esté en ejecución termine de utilizarlos.

Para ello usaremos regiones críticas, aunque sería posible coordinar los *threads* usando otros mecanismos más o menos avanzados (como los semáforos).

VENTAJAS

Es frecuente, dentro de la programación concurrente, que varios procesos compartan una misma variable. Se debe evitar, por consistencia, que mientras un proceso esté accediendo a una variable otro la esté modificando al mismo tiempo. Es decir, debemos conseguir la exclusión mutua de los procesos respecto a la variable compartida.

Existen varios mecanismos para proteger el acceso a recursos

Un mecanismo para conseguir la exclusión mutua es el uso de regiones críticas (en adelante, *RC*). El acceso a la variable compartida debe realizarse desde dentro de la *RC*, de tal forma que se deben cumplir una serie de condiciones:

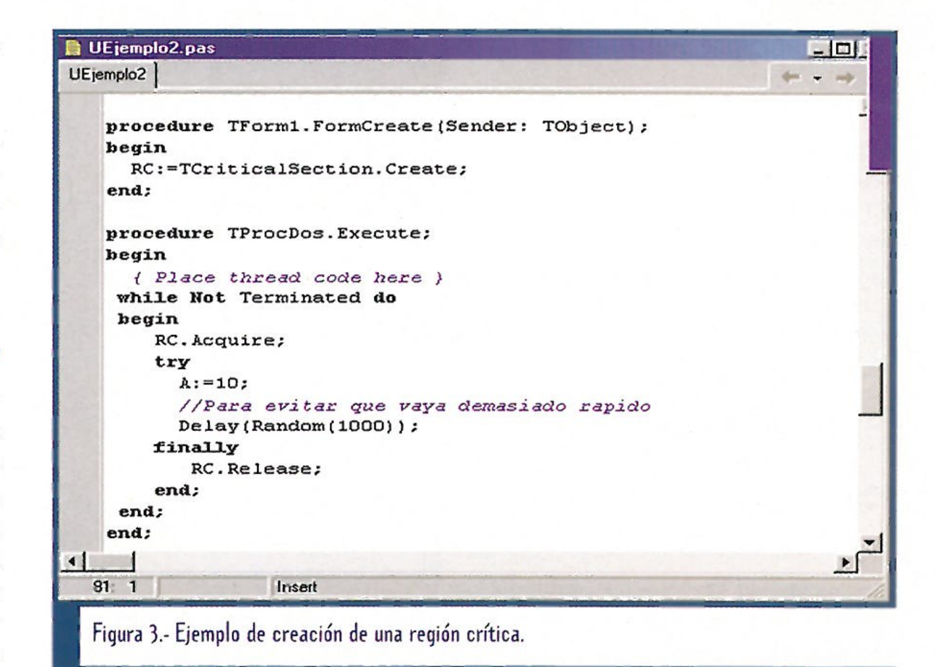


Figura 3.- Ejemplo de creación de una región crítica.

- Los procesos concurrentes sólo pueden acceder a las variables compartidas dentro de sus correspondientes *RC's*. Es decir, si compartimos la variable **v**, TODOS los *threads* que la utilicen deben acceder a ella desde el interior de una *RC*. Por ejemplo, supongamos que tenemos un cruce de dos carreteras, en la que hay cuatro coches esperando a cambiar de vía. Si todos respetan el orden de preferencia, no habrá problemas, pero si uno no lo hace es muy probable que ocurra un accidente.

- Un proceso que quiera entrar en una *RC* lo hará en un tiempo finito. Esto implica que el tiempo que un proceso debe permanecer dentro de una *RC* debe ser el menor posible, para que otros puedan acceder al recurso y no se reduzca el rendimiento debido a bloqueos innecesarios.

- Sólo puede haber un proceso a la vez en la misma *RC* en un instante de tiempo. Así aseguramos la exclusión mutua de procesos.

- Un proceso está dentro de una *RC* un tiempo finito, al cabo del cual la abandona. Conclusión: ¡¡ cuidado con los bucles infinitos!!

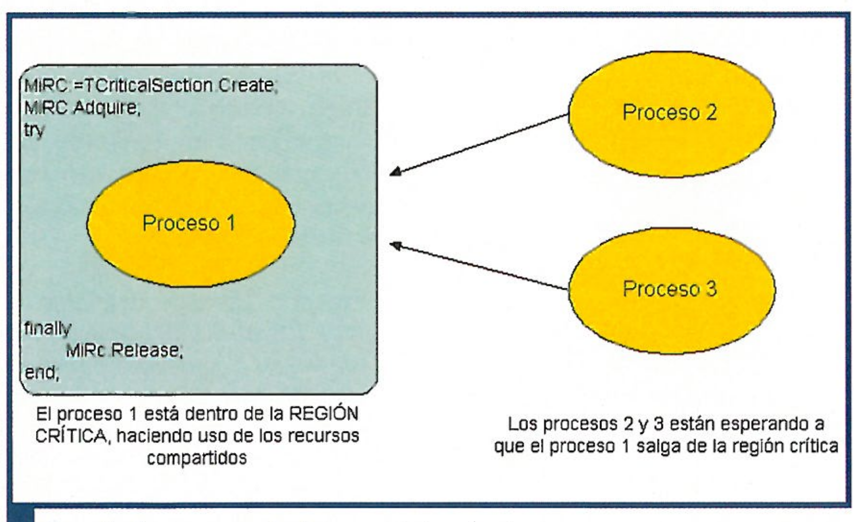


Figura 2.- Un proceso accede al recurso mientras el resto espera.

¿CUÁNDO NO SON NECESARIAS LAS RC'S?

Algunas clases tienen sus propios mecanismos de bloqueo, como por ejemplo la clase *TCanvas*. Esta clase dispone de los métodos *Lock* y *Unlock,* que permiten bloquear el acceso a otros threads a la superficie de dibujo.

Con algunas clases no es necesario usar RC's

La clase *TThreadList* permite crear listas de objetos a las que se puede acceder de forma segura desde otros *threads.* Para ello cuenta con los métodos *LockList* y *Unlocklist,* los cuales pueden ser anidados sin problemas.

El acceso al objeto no será liberado hasta que el último bloque *lock-unlock* se ejecute dentro del mismo *thread.*

CREACIÓN DE RC'S

Para crear una *RC* es necesario crear una instancia de la clase *TCriticalSection*. Después, mediante el método *Adquire* entraremos en la *RC*. Hasta que no llamemos al método *Release,* ningún otro *thread* podrá acceder al recurso al mismo tiempo que el thread en ejecución.

La mayoría de los accesos a un recurso compartido son de lectura

Es muy importante incluir la entrada y salida de la *RC* dentro de un bloque *try..finally,* para asegurar que el thread, pase lo que pase, siempre saldrá de la *RC*. Si no lo hiciésemos, y debido a alguna excepción no se ejecutase el método *Release,* bloquearíamos a todos los threads que estuviesen esperando para entrar en la RC.

```
Procedure TmiThread.Execute;
Begin

 //esperamos a que el otro
 //thread finalice

 If MiOtroThread.WaitFor=1 then
        HacerCosas;

end;
```

MiThread. Create
MiOtroThread. WaitFor
HacerCosas
Fin de MiThread
MiOtroThread. Create
MiOtroThread. Execute
Fin de MiOtroThread

Figura 5.- Este código produce un interbloqueo.

UNA ALTERNATIVA A LAS RC'S

Cuando se usan *RC's* se bloquea el acceso a la memoria, y ningún *thread,* excepto el que obtuvo el acceso a la región crítica, puede escribir o leer de ella. En la mayoría de los casos, los accesos a un recurso suelen ser de sólo lectura. Es posible que todos los *threads* lean de un recurso a la vez. Los problemas sólo se presentan cuando se intenta escribir sobre el recurso.

Puesto que las lecturas no implican ningún problema, es posible que bloqueando todo acceso estemos haciendo que el rendimiento del sistema no sea el adecuado.

En estos casos, conviene usar la clase *TMultiReadExclusiveWrite Synchronizer* para proteger el acceso a los recursos. Actúa como una *RC,* pero permite a múltiples threads leer de la memoria que protege sin problemas, siempre y cuando no haya ningún *thread* escribiendo.

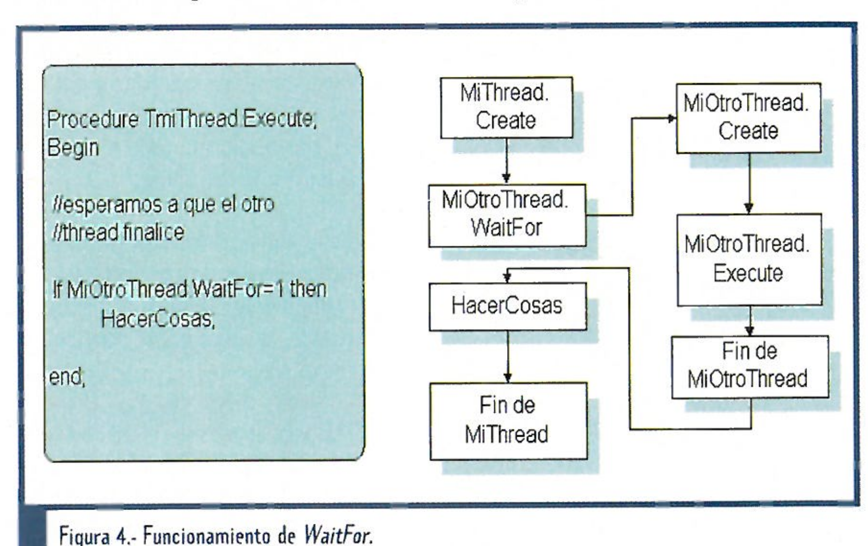


Figura 4.- Funcionamiento de *WaitFor.*

Su uso es parecido al de las *RC's*, de tal forma que cada *thread* que quiera leer del recurso debe llamar al método *BeginRead*. Este método se asegura de que no exista ningún *thread* que esté escribiendo en el recurso (en ese caso, se bloquearía la ejecución del thread). Al finalizar la lectura, se llama al método *EndRead*.

Es posible mejorar el rendimiento cuando los procesos de lectura son mayoritarios

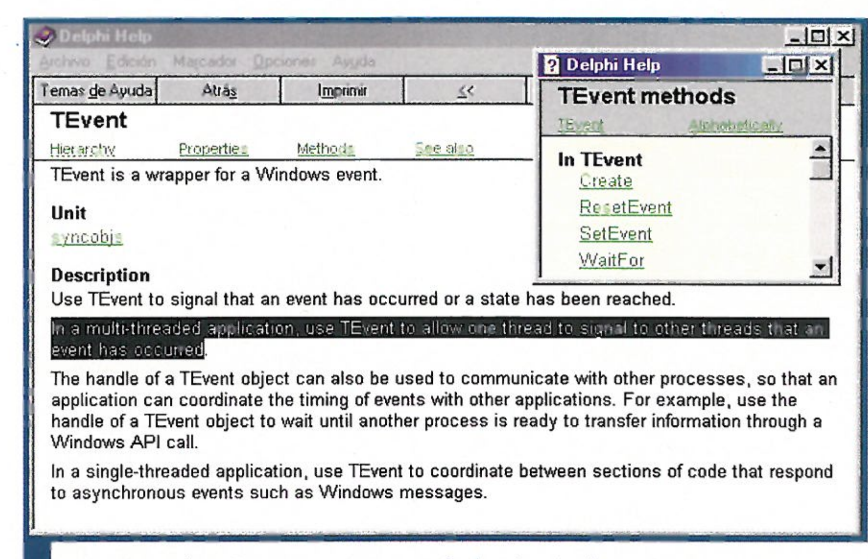


Figura 6.- La clase *TEvent* permite avisar a múltiples *threads* a la vez.

Para escribir en el recurso, primero se llama al método *BeginWrite*, que comprueba que no haya otro *thread* escribiendo o leyendo del recurso. Cuando la escritura finaliza, se llama al método *EndWrite*.

Al igual que las *RC's*, para que esta forma de protección funcione, todos los *threads* deben hacer uso

de ella. Para más información, consulte la ayuda de *Delphi*.

SINCRONIZAR THREADS

A veces, para que un *thread* pueda continuar, es necesario que otro finalice el trabajo que esté realizando. Ante esta situación tenemos:

- Esperar a que el otro *thread* termine su ejecución por completo.
- Esperar a que envíe una señal indicando que se ha finalizado la tarea por la que esperábamos (lo que no implica que haya finalizado su ejecución).

MÉTODO WAITFOR

Mediante el método *WaitFor* podemos hacer que un *thread* espere a que otro finalice.

En el ejemplo que se muestra en la Figura 5, hasta que no finalice la ejecución del thread *MiOtroThread* no se ejecutará el procedimiento *HacerCosas* (dentro del método *Execute* de la clase *MiThread).*

El método *WaitFor* devuelve un resultado cuyo valor es asignado por el método *Execute* del *thread* al que pertenece el método *WaitFor* (en el ejemplo, sería asignado por el método *Execute* del *thread MiOtroThread)*. Esto se realiza asignando un valor a la propiedad *ReturnValue* (de tipo entero).

Muy importante: NUNCA se debe llamar al método *WaitFor* de otro *thread* desde dentro de un método *Synchronize.* Podría provocar un bloqueo del sistema (aunque *Delphi* se daría cuenta de esto y lanzaría una excepción).

Observe la Figura 5. Si se ejecuta primero el *thread UnThread,* se llamaría al procedimiento *Prueba,* que lo primero que hace es esperar a que *OtroThread* finalice. ¡Pero *OtroThread* nunca podrá finalizar, porque a su vez está esperando a que *UnThread* salga del hilo principal!. Esta situación se denomina Interbloqueo.

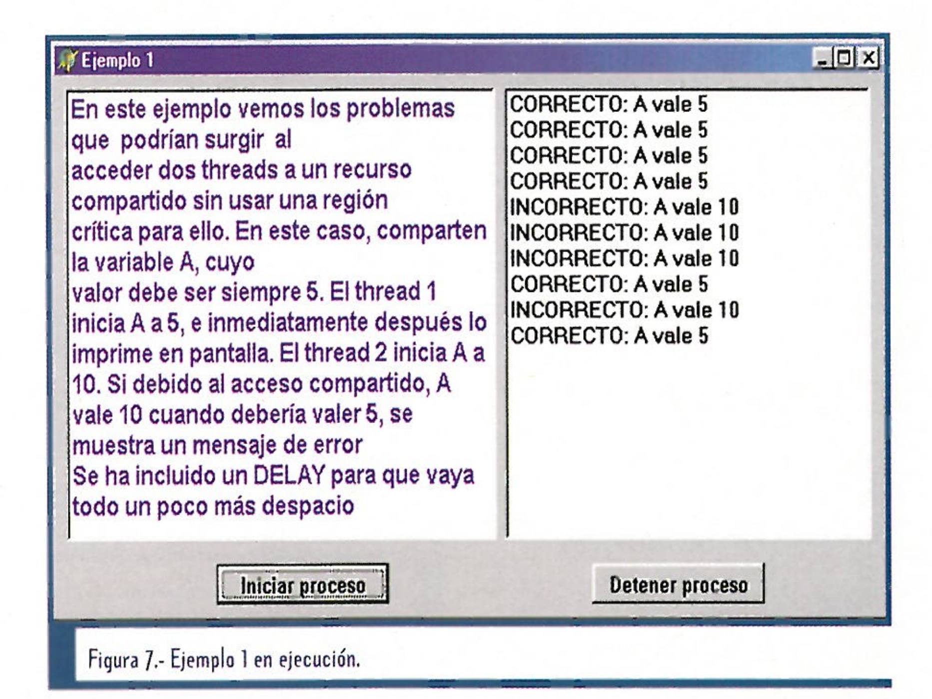


Figura 7.- Ejemplo 1 en ejecución.

LISTADO 1.

```
procedure TForm1.Iniciar;
begin
 //Iniciamos la variable evento
 Ev:=TEvent.Create(nil, true, false,'') ;
 //Iniciamos el contador
 Contador:=2;
 //Creamos los threads
 ProcUno:=TProceso.Create(false);
 ProcDos:=TProceso.Create(false);
 If (Ev.WaitFor(30000)=wrSignaled)
 then
    HacerCosas;
End;

procedure TProceso.Execute;
begin
  HacerCosas;
  DecContador;
end;
```

USO DE TEVENT

A veces es necesario esperar a que un *thread* finalice una determinada tarea, y no a que finalice su ejecución. Para esto se utiliza la clase *TEvent.* El método consiste en crear un objeto de tipo *TEvent* global a todos los threads.

Con SetEvent avisamos de la finalización de una tarea

Cuando un *thread* completa una operación, llama al método *SetEvent.* De esta forma los otros *threads* recibirán el evento, y sabrán que la tarea ha finalizado. Mediante el método *ResetEvent* se desactiva la señal.



TABLA 1. Ejemplos incluidos en el CD-ROM.

Ejemplo	Descripción
Ejemplo1.dpr	Muestra los problemas que pueden surgir cuando se comparten variables sin usar regiones críticas.
Ejemplo2.dpr	Muestra cómo los problemas del ejemplo 1 se solucionan usando regiones críticas.
Ejemplo3.dpr	Muestra como esperar a que otro proceso finalice usando el método WaitFor.
Ejemplo4.dpr	Muestra cómo esperar a que un número aleatorio de procesos finalicen para que otros procesos puedan seguir ejecutándose.

Por ejemplo, puede ser que necesitemos esperar a que una serie de *threads* finalicen. Como no sabemos cuál de los *threads* va a finalizar el último, no podemos usar el método *WaitFor.* Pero es posible usar un contador que indique cuántos *threads* quedan activos. Cuando este contador llegue a cero, se enviará una señal que indicará que todos los threads han finalizado.

Este es el código que cada *thread* debería ejecutar antes de finalizar su ejecución. La variable *RC* es de tipo *TCriticalSection*.

```
Procedure DecContador;
Begin
 RC.Adquirir;
 try
  Dec(Contador);
  If Contador=0 then
    Evento.SetEvent;
 finally
  RC.Release;
 End;
End;
```

El hilo principal inicia el contador, lanza los *threads,* y espera hasta que todos acaben. Para ello, ejecuta un *Evento.WaitFor.* Este método detiene la ejecución de un proceso hasta que reciba un evento (generado con *SetEvent).* Los posibles valores que puede devolver *WaitFor* son:

- *wrSignaled:* todo correcto.
- *wrTimeout:* se sobrepasó el tiempo limite sin haberse recibido la señal.
- *wrAbandoned:* el objeto Evento ha sido destruido antes de que pasase el periodo de TimeOut.
- *wrError:* ha ocurrido un error mientras se esperaba.

El código del Listado 1 muestra cómo el hilo principal lanza a los otros *threads* y espera a que todos finalicen.

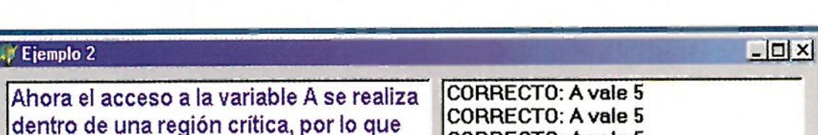


Figura 8.- Ejemplo 2 en ejecución.

EJEMPLOS

En la Tabla 1 se muestran las descripciones asociadas a cada uno de los ejemplos incluidos en el *CD-ROM*.

CONCLUSIÓN

Hemos visto algunos de los mecanismos que *Delphi* proporciona para sincronizar *threads.* El próximo mes veremos cómo acceder a bases de datos desde el interior de un *thread,* y pondremos algunos ejemplos de creación de *threads* usando directamente la *API* de *Windows.*



USCRIPCIÓN DOBLE

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO
SÓLO PROGRAMADORES

Sólo PROGRAMADORES LINUX

QUE NO TE LÍEN CON EL <CÓDIGO>



BOLETÍN DE SUSCRIPCIÓN

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L.
C/ San Sotero, 5. 1ª Planta. 28037 Madrid. Tlf: 91 304 87 64. Fax: 91 327 13 03

Quiero suscribirme a la revistas SÓLO PROGRAMADORES y SÓLO PROGRAMADORES LINUX desde el Nº.......................................y beneficiarme de las condiciones de estas magníficas promociones:

❑ **SUSCRIPCIÓN ANUAL**
24 NÚMEROS + 24 CD-ROMs
AL PRECIO DE
14.785 ptas. / 88,86 €

❑ **SUSCRIPCIÓN ANUAL ESPECIAL ESTUDIANTES**
24 NÚMEROS + 24 CD-ROMs
por sólo 11.830 ptas. / 71,01 €

FORMAS DE PAGO:

- ❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
- ❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41. Nº c/c: 0075/1040/43/ 0600047439
- ❑ Talón bancario a nombre de REVISTAS PROFESIONALES, S.L
- ❑ Domiciliación bancaria
- ❑ Contra reembolso

Soy antiguo suscriptor
❑ Sí ❑ No

PARA ENVÍOS AL EXTRANJERO SÓLO SE ADMITIRÁN LAS SIGUIENTES FORMAS DE PAGO:

- ❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
- ❑ Transferencia al Banco Popular Español. C/ Valdecanillas, 41. Nº c/c: 0075/1040/43/ 0600047439
- ❑ Eurocheque conformado con un banco español a nombre de REVISTAS PROFESIONALES, S.L

NOMBRE Y APELLIDOS:...

EDAD:................ PROFESIÓN: ....

TFNO: .... DOMICILIO: ....

CIUDAD:.. C.P: .... PROVINCIA:...............................

Promoción válida hasta agotar existencias

Datos de domiciliación:

Banco...

Domicilio ....

Nº de Cuenta.....................|.....................|.....................|.....................

Titular...

Fecha ....

FIRMA



Programar un reproductor MM con Delphi (y III)

Juan Luis Ceada Ramos
Programador en ARCABE Formación y Servicios Informáticos.

En esta entrega finalizaremos la implementación del reproductor multimedia que hemos estado desarrollando y veremos algunas nociones del estándar *MCI*.

INTRODUCCIÓN

Al final de la pasada entrega vimos que sería bastante útil poder tener un icono en la barra de tareas, junto al reloj del sistema (*tray icon*). De esta forma, nuestra aplicación, al ser minimizada, no aparecería en la barra de tareas, dejando el espacio libre para otras aplicaciones.

Además de la reducción de espacio, ganamos en comodidad, puesto que podemos asociar un menú a dicho icono que permita realizar las acciones más frecuentes sin tener que restaurar la aplicación.

IMPLEMENTACIÓN

Básicamente, el proceso de creación de un *tray icon* se resume en cuatro pasos tal y como veremos a continuación.

PASO 1

Introduciremos un componente *TPopUp* al que llamaremos *PopUp*. Crearemos las opciones de menú que aparecen en la Figura 2. Es posible insertar todas las opciones que deseemos (incluso submenús), pero las que se muestran en la figura son las básicas. A cada una de las opciones les asignaremos el mismo código que al botón correspondiente del formulario principal. Por ejemplo, a la opción *play* le corresponde el siguiente código:

```
procedure
    TFReproductor.Play1Click(Sen-
    der: TObject);
begin
    //código asociado al botón
    play
    PlayBtnClick(nil);
end;
```

La opción *Cerrar* ejecuta un simple *Close* y la opción *Restaurar Application. Restore*;

PASO 2

Es necesario definir una nueva variable, de tipo *TNtifyIconData*. Este tipo de datos se encuentra definido en la unidad *ShellApi*, la cual debemos incluir en la cláusula *uses* de la interfaz de la aplicación (hasta ahora estaba incluida, pero en la zona de implementación).

Nuestro reproductor dispondrá de su propio tray icon

El nombre de esta variable será *IconData* y la definiremos en la sección *private*. El momento indicado para rellenar la estructura *TNotifyIcon* es durante creación del formulario. En la Figura 3 se muestra el código necesario.

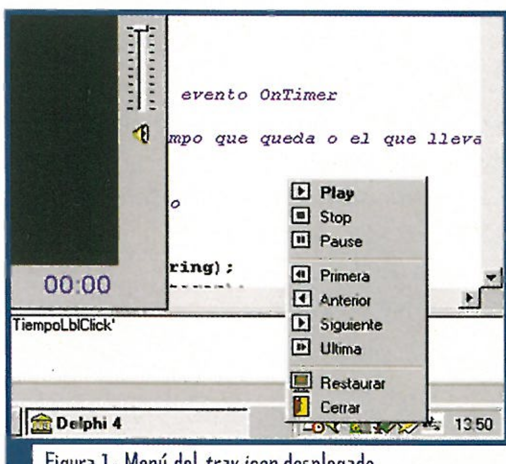


Figura 1.- Menú del *tray icon* desplegado.



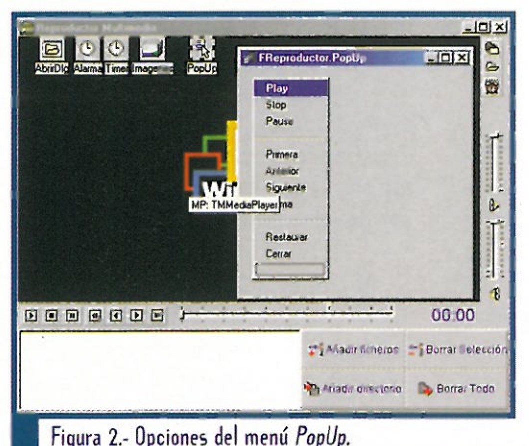


Figura 2.- Opciones del menú *PopUp*.

En la Tabla 1 es posible apreciar todos los campos que componen la estructura *TNotifyIcon*. Como podemos observar en la Figura 2, al campo *hWnd* le asignamos el *Handle* del formulario principal. Como identificador del icono, le asignamos el número 100. Por otra parte, el campo *uFlags* es una combinación de los tres valores que admite, lo que indica que el resto de los campos son válidos.

Para crear el tray icon, necesitamos una variable de tipo TNotifyIcon

Al campo *uCallbackMessage* le asociamos el código del mensaje que queremos que el sistema nos envíe cuando se pulse sobre el *tray icon*. En este caso, el mensaje que se deberá enviar tendrá como código *WM_USER + 1* (donde *WM_USER* es una constante que proporciona *Delphi* para saber a partir de qué número puede el usuario crear sus propios mensajes).

Por último, a los campos *hIcon* y *szTip* se les asocia el icono de la aplicación y el título (**Projects->Options->Application title**) respectivamente.

Sólo queda indicar al sistema que cree y muestre el *tray icon*. Para ello, basta con llamar a la función *Shell_NotifyIcon,* que recibe como parámetro la operación que realizar sobre el icono, y un puntero a la estructura que contiene su definición.

Las operaciones que se pueden realizar mediante esta función son: añadir (*NIM_ADD*), modificar *NIM_MODIFY*) y borrar (*NIM_DELETE*) un icono. Precisamente haremos uso de esta última opción cuando cerremos la aplicación (evento *OnDestroy*), para hacer que el *tray icon* desaparezca de la barra de tareas.

PASO 3

Nuestra aplicación debe procesar los mensajes de tipo *WM_USER + 1*, que son los que el sistema envía cuando hacemos *clic* sobre el *tray icon* en la barra de tareas. Para ello habrá que redefinir el procedimiento *WndProc*.

Todas las aplicaciones que creemos tienen un procedimiento *WndProc*, puesto que este es el encargado de procesar los mensajes que recibe la aplicación. La mayoría de las veces no se necesita trabajar con él, excepto cuando hay que tratar mensajes definidos por el propio programador.

En la Figura 4 se muestra el código de dicho procedimiento. Como se puede observar, comprobamos el tipo de mensaje que recibimos. Si es de tipo *WM_USER + 1*, vemos qué botón se ha pulsado (viene indicado en el campo *lParam* del mensaje). Si es el botón derecho del ratón, obtenemos las coordenadas del cursor y hacemos que aparezca el menú *popup* en dichas coordenadas.

Por último, pasamos el control al manejador de mensajes por defecto (mediante la sentencia *inherited*).

PASO 4

Para finalizar tendremos que añadir código al evento *OnShow* del formulario, y al *OnMinimize* de la aplicación. Con esto conseguiremos que al iniciarse la aplicación, ésta no aparezca en la barra de tareas. Además, cada vez que la restauremos y la volvamos a minimizar, volverá a desaparecer de la barra de tareas (esto lo controla el evento *OnMinimize*). Las siguientes líneas de código realizan todo el proceso:

```
//evento OnCreate
Application.OnMinimize:=Ocultar;
//evento OnShow
Ocultar(nil);
procedure
      TFReproductor.Ocultar(Sender:
      TObject);
```

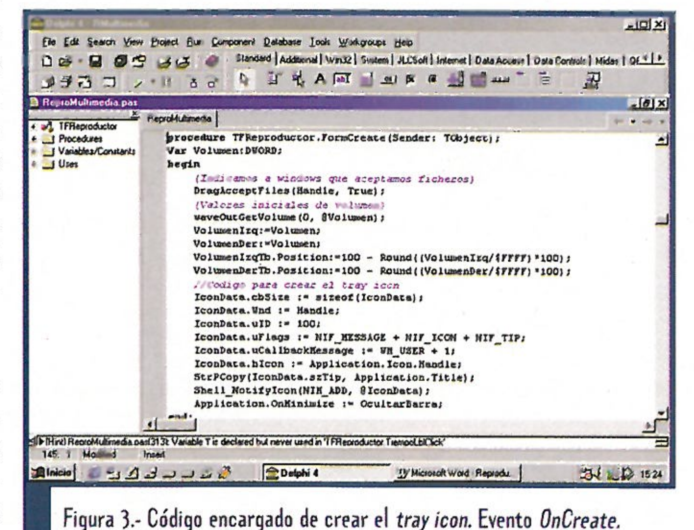


Figura 3.- Código encargado de crear el *tray icon*. Evento *OnCreate*.

```
begin
  ShowWindow(Application.Handle,
    SW_HIDE);
end;
```

Con esto damos por terminada la implementación del reproductor multimedia. Es susceptible de muchas mejoras, pero esto ya queda en manos de los lectores. Veamos ahora algunas nociones acerca del estándar *MCI*.

EL ESTANDAR MCI

El estándar *MCI* (*Media Control Interface*) proporciona una serie de comandos que son parte de una interfaz genérica que permite controlar cualquier clase de dispositivo multimedia. Al igual que el *API DirectX*, permite a los desarrolladores centrarse en la programación de aplicaciones, sin tener que preocuparse de cómo funciona internamente cada dispositivo.

Este estándar proporciona a las aplicaciones la capacidad de controlar periféricos de audio y vídeo (como por ejemplo, tarjetas de sonido, *CD-Audio*, vídeos digitales, secuenciadores *MIDI*, etc.).

TIPOS DE COMANDOS

Para controlar los periféricos podemos trabajar con mensajes o cadenas, de tal forma que en ambos casos podemos efectuar las mismas operaciones. Sólo varía el formato de los comandos. Por ejemplo, las dos siguientes líneas sirven para realizar la acción *play*. La primera envía la orden usando un mensaje y la segunda mediante una cadena:

```
mciSendCommand( dis-
     positivo,
     MCI_PLAY,
     flags,

     Longint(@para-
     metros));
mciSendString('play
     dispositivo
     parametros
     flags',
                 nil,
     0, nil)
```

Aunque no son excluyentes, nos centraremos en la primera opción, es decir, usaremos mensajes para comunicar las órdenes a los dispositivos. El uso de mensajes simplifica la programación, ya que devuelve los resultados en estructuras de datos fáciles de interpretar en cualquier lenguaje de programación.

Usando cadenas, aunque es más fácil recordar los comandos disponibles, se complica la forma de obtener información de los dispositivos, ya que el resultado de las operaciones se devuelve en forma de cadenas que hay que analizar (mediante un *parser*) para extraer resultados.

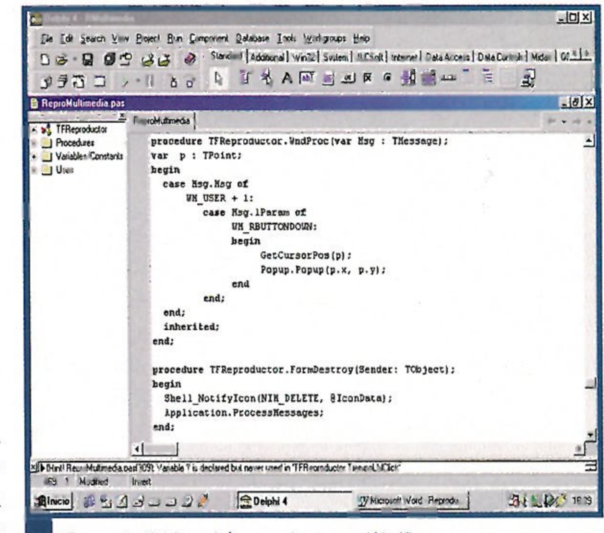


Figura 4.- Código del procedimiento *WndProc*.

TABLA 1. Estructura TNotifIcon.

Campo	Descripción
CbSize: DWORD	Tamaño de la estructura.
HWnd: Thandle	Handle de la ventana que recibe los mensajes asociados con El Tray icon.
UID: Integer	Número que identifica al icono. Puede ser el que nosotros queramos.
Uflags: Integer	Indica cuáles de los siguientes campos son válidos. Puede ser una combinación de estos valores:
	NIF_ICON: el campo hIcon es válido. NIF_MESSAGE: el campo uCallbackMessage es válido. NIF_TIP: el campo szTip es válido.
UcallbackMessage: Integer	Indica qué mensaje enviará el tray icon cuando se pulse sobre él.
Hicon: Thandle	Handle al bitmap que contiene el icono.
SzTip: String	Hint que se mostrará al situar el cursor sobre el icono.



TABLA 2. Algunos de los mensajes disponibles en cada grupo.

M. Sistema	Descripción	
MCI_BREAK	Establece la combinación de teclas que permite abortar cualquier operación.	
MCI_SYSINFO	Devuelve información sobre los drivers MCI.	
M. Requeridos	**Descripción**	
MCI_GETDEVCAPS	Devuelve las capacidades de un dispositivo.	
MCI_CLOS	ECierra el dispositivo.	
MCI_OPEN	Inicia el dispositivo.	
MCI_STATUS	Devuelve información sobre el estado de los dispositivos.	
M. Opcionales	**Descripción**	
MCI_LOAD	Carga datos de un fichero.	
MCI_PAUSE	Pausa la reproducción.	
MCI_PLAY	Comienza la reproducción.	
MCI_SEEK	Moverse dentro del medio reproducido.	
MCI_SET	Establecer opciones del dispositivo.	
MCI_STOP	Detiene la reproducción.	
M. Extendidos	**Aplicable a**	**Descripción**
MCI_SETAUDIO	Digital vídeo	Parámetros de audio del vídeo.
MCI_SETVIDEO	Digital vídeo	Parámetros de vídeo.
MCI_WINDOW	Digital vídeo, overlay	Controla la ventana de visualización.

CLASIFICACION DE LOS COMANDOS MCI

La interfaz *MCI* se divide en cuatro partes:

- **Comandos del sistema:** son controlados por *MCI* directamente, en lugar de por el *driver* del dispositivo.
- **Comandos requeridos:** son controlados por el *driver*. Todos los *drivers MCI* deben soportar este subconjunto de comandos.
- **Comandos opcionales:** sólo son soportados por algunos dispositivos.
- **Comandos extendidos:** son específicos de un determinado tipo de dispositivo.

Cualquier *driver MCI* debe soportar los comandos de sistema y los comandos requeridos. Sin embargo, los comandos opcionales y los extendidos no son soportados por todos los *drivers*. En nuestras aplicaciones podremos usar cualquier mensaje que pertenezca a los dos primeros grupos sin problemas, pero para usar los del grupo tres y cuatro sería conveniente consultar al dispositivo cuáles son sus capacidades, y por lo tanto, qué mensajes soporta.

En la Tabla 2 se muestra un pequeño resumen de los mensajes que se incluyen en cada uno de los grupos. Existen muchos más, por lo que remito al lector al fichero de ayuda *mmedia.hlp*, o a las direcciones *Web* que se indican al final del artículo para más información.

TRES FLAGS

La mayoría de los comandos *MCI* incluyen tres *flags* que modifican el comportamiento del comando. Los *flags MCI_WAIT* y *MCI_NOTIFY* son comunes a todos los comandos. El *flag MCI_TEST* sólo está disponible cuando el dispositivo que tratar sea un vídeo digital o un *videodisc*, y sirve para preguntar al dispositivo si puede realizar o no una determinada operación.

El *flag MCI_WAIT* indica al dispositivo que no devuelva el control

a la aplicación hasta que la operación que se le ha indicado no se haya realizado por completo. Es decir, este *flag* permite indicar si un determinado comando se ejecuta de forma bloqueante o no.

El *flag MCI_NOTIFY* indica al dispositivo que debe generar un mensaje de tipo *MM_MCINOTIFY* cuando complete la operación que se le ha ordenado realizar. El mensaje indica que la operación indicada finalizó y además si se completó con éxito, o se abortó, o falló por algún motivo. Será el usuario, en un procedimiento de tratamiento de mensajes, el que compruebe cuál fue el resultado de la última operación. Vea el ejemplo contenido en el *CD* para más información.

DISPOSITIVOS SOPORTADOS

El estándar *MCI* reconoce una serie de dispositivos básicos. En la Tabla 3 se muestran todos los dispositivos soportados. La mayoría de los comandos requieren que se les pase como parámetro el tipo de dispositivo que usar. Por ejemplo, si queremos iniciar un *CD* audio, habrá que hacer lo siguiente:

```
OpenParm.lpstrDeviceType :=
     'cdaudio';
Flags:=mci_Open_Type;
mciSendCommand(0, MCI_OPEN,
     Flags, Longint(@OpenParm));
```

Nuestra aplicación de ejemplo trabaja con *CD's* de audio, y ficheros de sonido/vídeo. En este último caso, no se les indica a los comandos el dispositivo que deben usar para reproducirlos. En estos casos se hace la búsqueda por nombre. Es decir, en base a la extensión del fichero que reproducir, el sistema se encargará de seleccionar un *driver* u otro.

En los compuestos, se trabaja con archivos que pueden ser creados/modificados

Puesto que la definición del estándar es algo antigua, no existen constantes que identifiquen los dispositivos encargados de reproducir, por ejemplo, ficheros *MP3* o *MPEG*, estando éstos están englobados dentro del tipo de dispositivo *OTHER*. Es por esto que lo mejor es indicar al sistema el nombre del fichero que reproducir, y que él sea el encargado de seleccionar el *driver* adecuado para reproducirlo.

Es esencial averiguar qué comandos soporta el dispositivo antes de proceder

Los diferentes dispositivos se dividen en dos tipos: compuestos y simples. Los primeros requieren archivos en los cuales se almacenan los datos, ya sean sonidos digitalizados, vídeos, etc. Al ser archivos, es posible su modificación/creación. Los segundos no requieren archivos. Los datos están almacenados en un medio físico, ya sea un *CD-Audio*, una cinta de vídeo, etc.

DRIVERS

Los *drivers* proporcionan toda la funcionalidad a cada uno de los comandos *MCI*. Toda lo relacionado con la reproducción, representación y grabación en dispositivos multimedia es controlado por los *drivers*. Cada *driver* varía en el

TABLA 3. Dispositivos soportados por el estándar MCI.

Dispositivo	Descripción
Avivideo	Archivo vídeo for Windows (compuesto).
Cdaudio	CD audio (simple).
Dat	Digital-audio tape (simple).
Digitalvideo	Vídeo digital en ventana (sin usar el GDI) (simple).
Other	Dispositivo indefinido.
Overlay	Vídeo analógico en ventana (simple).
Scanner	Scanner (simple).
Sequencer	Secuenciador MIDI (compuesto).
Vcr	Vídeo (simple).
Videodisc	Vídeodisc (simple).
Waveaudio	Dispositivo de audio (compuesto).



número de comandos y *flags* soportados del estándar *MCI*.

Con el mensaje MCI_OPEN, el sistema analiza los parámetros que van unidos a dicho mensaje

Por ejemplo, el comando *MCI_RECORD* forma parte del conjunto de órdenes de un secuenciador *MIDI*, pero el driver *MCISEQ* incluido con *Windows* no ofrece soporte para este comando en particular. Aunque la especificación del estándar dice que el comando *MCI_RECORD* forma parte del conjunto de órdenes que soporta un secuenciador, no todos los dispositivos de este tipo tienen por qué reconocer dicho comando. Es lógico por tanto, que el *driver* proporcionado con *Windows*, al ser genérico, no dé soporte a este comando en particular.

Las aplicaciones deberían usar el comando *MCI_GETDEVCAPS* para averiguar qué soporta un dispositivo específico, y obrar en consecuencia.

Cuando enviamos el mensaje *MCI_OPEN*, el sistema analiza los parámetros que van unidos a dicho mensaje. En caso de que se indique directamente el dispositivo a abrir, *Windows* cargará en la memoria el *driver* adecuado, y a partir de ahí, pasará el control del dispositivo al *driver*.

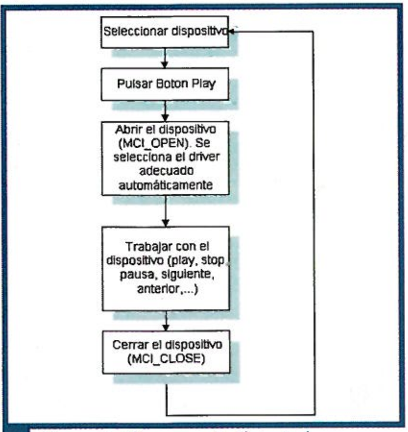


Figura 5.- Diagrama de flujo de la aplicación de ejemplo.

En caso de que no se indique un dispositivo en particular, sino que la carga de un *driver* u otro dependa de la extensión del fichero a reproducir, *Windows* buscará en la base de datos de ficheros registrados en el sistema cuál es el *driver* que debe cargar.

EJEMPLO

Una vez dadas unas nociones de cómo funciona y de cómo está implementado el estándar *MCI*, pasaremos a la parte práctica.

Para demostrar cómo se trabaja con la interfaz *MCI* se ha desarrollado una pequeña aplicación, cuyo código fuente, profusamente comentado, podéis encontrar en el *CD-ROM*.

En dicha aplicación se implementa, básicamente, el diagrama de flujo que se muestra en la Figura 5. Este diagrama representa de forma esquemática los pasos a seguir para trabajar con cualquier dispositivo multimedia.

Con este programa, podréis controlar un *CD-Audio*, reproducir ficheros de audio (*WAV, MP3's*, etc) y vídeos (*AVI, MPG, etc*). Básicamente, se puede decir que hemos implementado un "mini" *Windows Media Player*. Como podrá observar en el código, no se hace un tratamiento especial en función del tipo de dispositivo a utilizar. El código es el mismo en todos los casos.

Es una de las ventajas de usar la interfaz MCI. Permite controlar aplicaciones que funcionarán sobre un gran número de dispositivos sin que ello implique cambiar el código fuente.

Al igual que Windows Media Player, el ejemplo funciona gracias a MCI

Bueno, la verdad es que no somos independientes al cien por cien de los dispositivos con los que vamos a trabajar. Es necesario comprobar qué características soporta cada dispositivo en particular. Por ejemplo, puesto que la tarjeta de sonido no soporta la operación "expulsar", pues no tiene sentido que el botón de expulsión esté activo. Si deberá aparecer si controlamos un lector de *CD-ROM*, para poder abrir y cerrar la bandeja.

SABER MÁS

Además de terminar la programación de nuestro reproductor, en este artículo hemos visto las nociones básicas del estándar *MCI*. Los lectores que lo deseen podrán, combinando las nociones teóricas aquí expuestas y el ejemplo práctico, desarrollar nuevos proyectos sin ningún problema.

En las siguientes direcciones podrá obtener abundante información acerca del estándar *MCI*.

http://www.hauppauge.com/html/mci.htm
http://www.compguy.com/cwtip05.htm
http://msdn.microsoft.com/library/default.htm

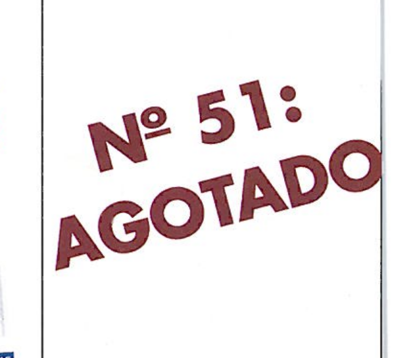
¿TE FALTA ALGÚN NÚMERO DE...?

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES





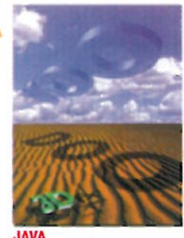


LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES



CONTENIDO DEL CD-ROM



DESARROLLO CORBA
Marcando la diferencia entre la norma CORBA y DCOM

HTML DINAMICO
El efecto Scrolling

INTERNET
Cómo crear un buscador

JAVA
Construcción de aplicaciones gráficas multiplataforma con AWT

SEGURIDAD
Un cortafuegos con FireWall Toolkit

MULTIMEDIA
Reconocimiento de voz: DictaPad

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

CREAR UN CHAT CON JAVA



CLIENTE Y SERVIDOR

NUEVAS TÉCNICAS DE BASES DE DATOS
Oracle, Informix, PostgreSQL

MULTIMEDIA
Reconocimiento de voz: E-mails

PROGRAMACIÓN GRÁFICA
Construir un Z buffer

HERRAMIENTAS
Novedades de Visual Basic 6.0

HTML DINÁMICO
Áreas desplegables

LINUX
Planificación de procesos

CONTENIDO DEL CD-ROM

LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

USO DE IMÁGENES con el API JAVA 2D

MULTIMEDIA
Reconocimiento de voz. Gestión telefónica

HERRAMIENTAS
VBScript y Active Server Pages

PROGRAMACIÓN GRÁFICA
Construcción de un polígono. Nuevas técnicas

COMUNICACIONES
Estudio práctico del TCP

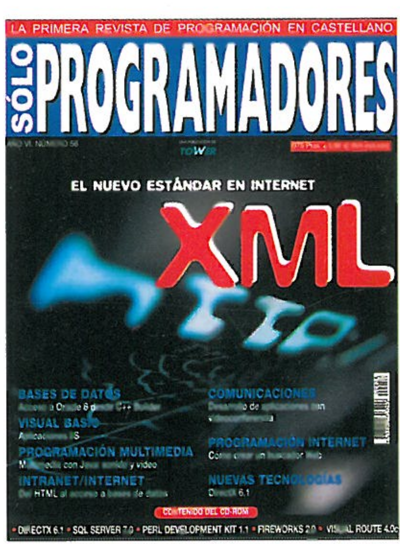
HTML DINÁMICO
Menús desplegables



CONTENIDO DEL CD-ROM















BOLETÍN DE PEDIDOS

Rellene o fotocopie el cupón y envíelo a REVISTAS PROFESIONALES, S.L.
(Revista Sólo Programadores). C/ San Sotero, 5. 1ª Planta. 28037 Madrid
Tlf: 91 304 87 64. Fax: 91 327 13 03

Deseo que me manden los número/s...............................
....
....

NOMBRE Y APELLIDOS:...
....

EDAD:.......... PROFESIÓN: ....TFNO: ....

DOMICILIO: ....

CIUDAD: ....

PROVINCIA: ....C.P: ....

❑ **Precio por unidad: 975 pesetas** + 700 ptas. de gastos de envío.

FORMAS DE PAGO:

❑ Giro postal a nombre de REVISTAS PROFESIONALES, S.L
❑ Talón bancario a nombre de REVISTAS PROFESIONALES. S.L
❑ Domiciliación bancaria
❑ Contra reembolso

DATOS DE DOMICILIACIÓN:

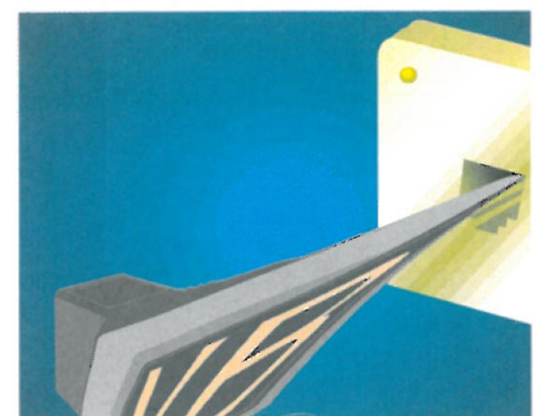
Banco.. FIRMA

Domicilio...

Nº de Cuenta..............|..............|..............|..............................

Titular..

Fecha...



TAPI: API de telefonía (y III)

Constantino Sánchez Ballesteros.
Técnico Superior de Desarrollo de Aplicaciones

Finalizamos esta serie sobre telefonía terminando el proyecto de *Visual Basic* que dejamos pendiente el pasado mes.

FORMULARIO PRINCIPAL DEL PROGRAMA (FRMTAPI.FRM)

Este formulario contiene el código principal del programa. Para cada botón del formulario se asocia un evento que realiza una tarea determinada. En la Figura 1 se puede apreciar el aspecto del formulario en ejecución.

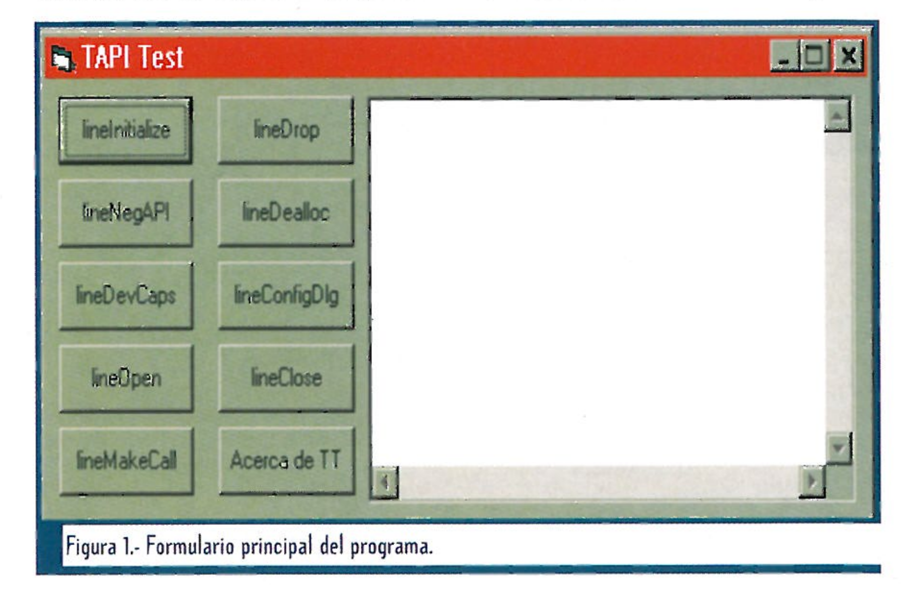


Figura 1.- Formulario principal del programa.

Vamos a seguir paso por paso el orden establecido en los botones del formulario a la hora de desglosar el código de los mismos. Primero empezaremos por el botón **lineInitialize**.

BOTÓN LINEINITIALIZE

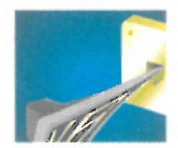
Se encarga de inicializar la línea mostrando el número de dispositivos de línea (*modems*) que tenemos instalados en nuestro equipo (normalmente uno) para su posible utilización.

Para realizar una llamada telefónica, es necesario inicializar la línea

Para inicializar la línea utilizamos la función *LineInitialize*. Una vez hecho esto obtendremos en la caja de diálogo la información relevante a la operación efectuada. En la Figura 2 podemos ver el resultado de esta función.

BOTÓN LINENEGAPI

Este botón ofrece información relativa a la versión soportada por el dispositivo anteriormente abierto. Si pulsamos sobre este botón antes de realizar la apertura de línea no aparecerá ninguna información en la caja de diálogo. En caso de haber logrado ejecutar con éxito la función (habiendo abierto la



línea anteriormente) se mostrará un *status* en la caja de diálogo mostrándonos información sobre la operación realizada (Figura 3).

La API permite saber qué y cuántos módems tenemos instalados

En el Listado 2 podemos ver el desarrollo del código encargado de realizar esta operación. Debemos prestar especial atención a la sintaxis de la función específica de *TAPI* para cada evento que codifiquemos, puesto que es la parte que nos interesa en este caso. En esta ocasión, la función encargada de establecer la negociación es *lineNegotiateAPIVersion.*

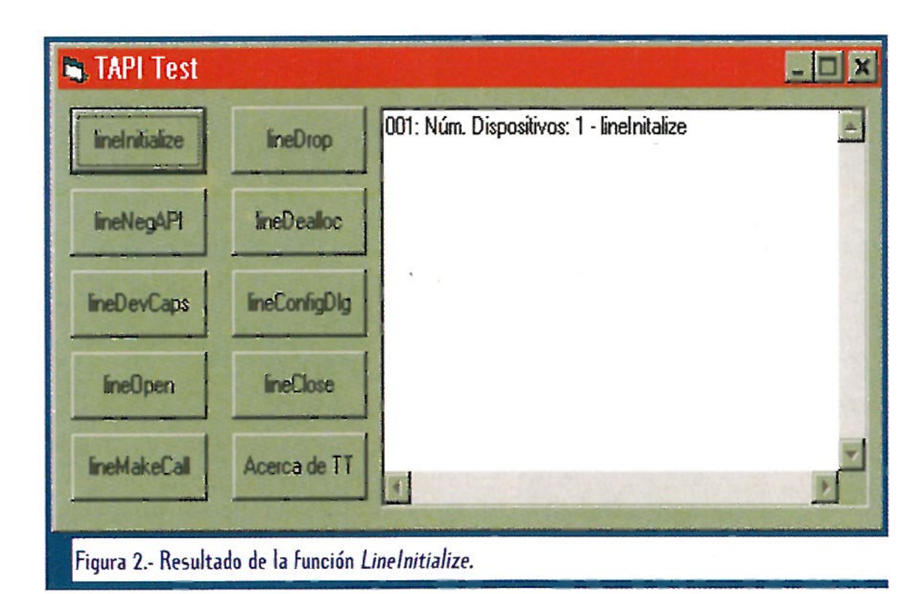


Figura 2.- Resultado de la función *LineInitialize.*

BOTÓN LINEDEVCAPS

La información relativa a las características del dispositivo que hará uso de la línea, que en este caso será el módem. En la Figura 4 se puede apreciar el detalle de lo que ocurre al pulsar sobre este botón.

Para que la función *lineGet DevCaps* haga su trabajo de manera correcta es necesario haber realizado anteriormente una inicialización de línea mediante el primer botón del formulario (**LineInitialize**). En el código siguiente se puede apreciar la programación del botón **LineGetDevCaps:**

```
Private Sub Command4_Click()
Dim a As Long
Dim b As LINEDEVCAPS
Dim x As Integer

For x = 0 To lines - 1
   b.dwTotalSize =
    LINEDEVCAPS_FIXEDSIZE + 2048

   a = lineGetDevCaps(lineApp, x,
    65540, &H0, b)

   If a > 0 Then
     AddText Me.Text1,
    TapiErrMsg(a) & " -
    lineGetDevCaps"
   Else
    AddText Me.Text1,
    GetVarInfo(Clean(b.mem),
       (b.dwLineNameOffset -
    LINEDEVCAPS_FIXEDSIZE) + 1,
       b.dwLineNameSize - 1) & "
    - Nombre línea"
```

LISTADO 1.

```
Private Sub Command1_Click()
Dim retcode As Long

hInst = 0
lineApp = 0
lines = 0

adrCallBack = PtrToLong(AddressOf LINECALLBACK)

hInst = GetModuleHandle(vbNullString)

retcode = lineInitialize(lineApp, hInst,
     adrCallBack, "TAPI OUT", lines)

If retcode <> 0 Then
   AddText Text1, TapiErrMsg(retcode) &
       " - lineInitalize"
Else
  AddText Text1, "Núm. Dispositivos: " &
       CStr(lines) & " - lineInitalize"
End If

End Sub
```

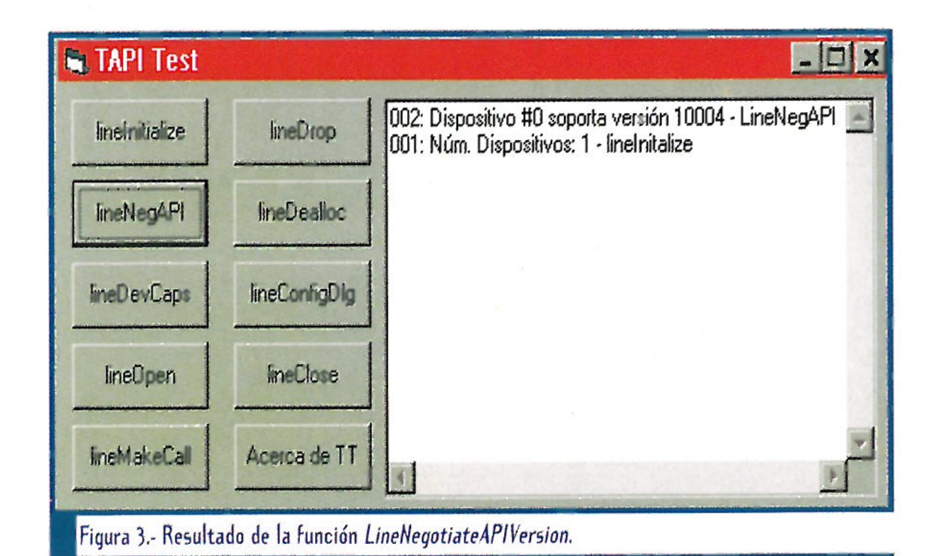


Figura 3.- Resultado de la función *LineNegotiateAPIVersion.*

```
        End If
Next
End Sub
```

Se utiliza un bucle *For/Next* para enumerar todos los dispositivos de línea (*modems*) que tengamos instalados en el sistema. La variable **a** indica si existe algún dispositivo instalado en el sistema (cuando sea mayor que **0**).

BOTÓN LINEOPEN

Hace algo tan "relativamente" sencillo como abrir una línea. Para ello utilizamos la función *LineOpen*, que debe ser previamente inicializada mediante el establecimiento de unas propiedades (método *dwTotalSize* de la variable **udtLineCall**). Como siempre, invito al lector a que le dé un repaso a la ayuda de la *API* para ver las posibilidades existentes.

La función *LineOpen* contiene varios parámetros importantes:

- *LINECALLPRIVILEGE_NONE*: Establecemos como privilegio de llamada "ninguno".

- *LINEMEDIAMODE_DATAMODEM*: El *Media Mode* se establece como módem/datos.

```
Private Sub Command5_Click()
Dim a As Long
  udtLineCall.dwTotalSize = LINE-
    CALLPARAMS_FIXEDSIZE + 2048
  a = lineOpen(lineApp, 0,
    lphLine, 65540, &H0,
    AddressOf LINECALLBACK, LINE-
    CALLPRIVILEGE_NONE,
    LINEMEDIAMODE_DATAMODEM,
    udtLineCall)
  If a <> 0 Then
    AddText Me.Text1,
    TapiErrMsg(a) & " - lineOpen"
   Else
     AddText Me.Text1,
    Hex(udtLineCall.dwMediaMode)
    & " - lineOpen"
    End If
End Sub
```

BOTÓN LINEMAKECALL

Este botón ejecutará la acción más importante que buscamos en el programa. Se trata de la creación de la llamada telefónica (véase Figura 5). Por supuesto, para poder llegar hasta aquí hemos tenido que haber realizado antes dos pasos primordiales:

- Inicialización de la línea telefónica (*LineInitialize*).

- Apertura de la línea (*LineOpen*).

Recordemos que el botón de negociación de *API* y el de caracte-

LISTADO 2.

```
Private Sub Command3_Click()

Dim a As Long
Dim x As Integer
Dim lAPIVer As Long
Dim linexID As LINEEXTENSIONID

For x = 0 To lines - 1
a = lineNegotiateAPIVersion(lineApp, x, 65536,
         65540, lAPIVer, linexID)
If a < 0 Then
   AddText Me.Text1, TapiErrMsg(a) &
          " - LineNegAPI(" & CStr(x) & ")"
Else
  AddText Me.Text1, "Dispositivo #" &
         CStr(x) & " soporta versión " & Hex(lAPIVer) &
         " - LineNegAPI"
End If
Next

End Sub
```



rísticas del dispositivo son meramente informativos, aunque tienen su utilidad dependiendo de los casos.

Las llamadas de voz son definidas por TAPI como "voz interactiva"

Para poder crear la llamada es necesario especificar de qué tipo es. Recordemos que podemos realizar varios tipos de llamadas: voz interactiva, datos, etc.

El procedimiento que sigue este botón es el siguiente: una vez pulsado el botón aparecerá una caja de diálogo en la que introduciremos el número que marcar. Por defecto, se ha establecido el número **1004**, que es el primer número que veremos en pantalla.

El siguiente paso consiste en establecer el *Media Mode* para la llamada. Como en nuestro caso va a ser una llamada de voz asignaremos al método *dwMediaMode* el valor **LINEMEDIAMODE_INTERACTIVEVOICE.**



Figura 4.- Resultado de la función *LineGetDevCaps.*

Por último, sólo queda hacer una llamada a la función *Line MakeCall* y se iniciará una nueva sesión telefónica automáticamente. Para cortar la conexión será necesario pulsar sobre el botón *LineClose*, que se encarga de cortar la comunicación cerrando la línea. A continuación se puede ver el código completo asignado al evento *Click* de este botón:

```
Private Sub Command6_Click()
Dim a As Long
Dim strAddr As String
strAddr = InputBox("Introduzca
    número a marcar:",
     "lineMakeCall", "1004")
If Trim(strAddr) = "" Then Exit
    Sub
```

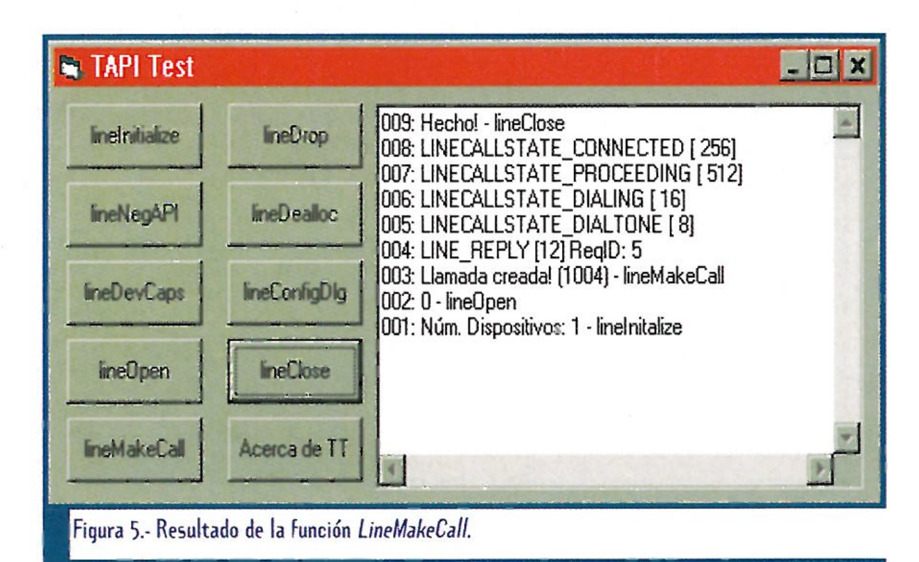


Figura 5.- Resultado de la función *LineMakeCall.*

```
  udtLineCall.dwMediaMode =
   LINEMEDIAMODE_INTERACTIVEVOI-
   CE
  a = lineMakeCall(lphLine,
   lphCall, strAddr, 0,
   udtLineCall)
 If a < 0 Then
    AddText Me.Text1,
   TapiErrMsg(a) & " -
   lineMakeCall"
 Else
    AddText Me.Text1, "Llamada
   creada! (" & strAddr & ") -
   lineMakeCall"
   End If
End Sub
```

BOTÓN LINECONFIGDLG

Este botón abrirá el cuadro de diálogo de *Windows* encargado de visualizar las propiedades del módem instalado en el ordenador (véase Figura 6). Para ello, será necesario introducir en el cuadro de diálogo el número de dispositivo que queremos configurar. Este número podemos saberlo cuando se ha pulsado el botón de negociación *API* (*LineNegAPI*). Por defecto, el **0** representa el primer módem detectado.

Un detalle importante se refiere al hecho de pasar en uno de los parámetros de la función *line ConfigDialog* la cadena de caracteres **comm.** De este modo se abrirá el cuadro de diálogo correcto. Si cambiamos este nombre, obtendremos un error en la ejecución de la función.

```
Private Sub
    cmdLineConfigDialog_Click()
 Dim retcode As Long
 Dim lngDev As Long
 lngDev = 0
 lngDev = InputBox("Introduzca
    ID# del dispositivo de
    línea:",
      "lineConfigDialog",
    lngDev)
 retcode =
    lineConfigDialog(lngDev,
    Me.hWnd, "comm")
 If retcode <> 0 Then
    AddText Me.Text1,
    TapiErrMsg(retcode) & "
    - lineConfigDialog"
 End If
End Sub
```

Los botones restantes tienen código importante para el programa pero poco significativo para ser comentado.

Por ejemplo, cuando se cierra o se libera una línea, debemos pasar como parámetro a la función correspondiente el *handle* de la línea o de la llamada creada (representado por las variables **lphLine** y **lphCall**).

```
Dim retcode As Long
retcode=lineDrop(lphCall, "", 0)
retcode=lineClose(lphLine)
retcode=lineDeallocateCall
     (lphCall)
```

CONCLUSIÓN

Con esto finalizamos esta interesante serie relacionada con el mundo de la telefonía mediante nuestro *PC*. Como consejo, aunque sea obvio, remito al lector encarecidamente a que eche un vistazo a la ayuda de la *API*, además de los ejemplos incluidos.

Aunque la ayuda técnica pueda venir para *C++*, los usuarios de *Visual Basic* no tendrán problemas en acoplar los conceptos importantes a su lenguaje de programación.

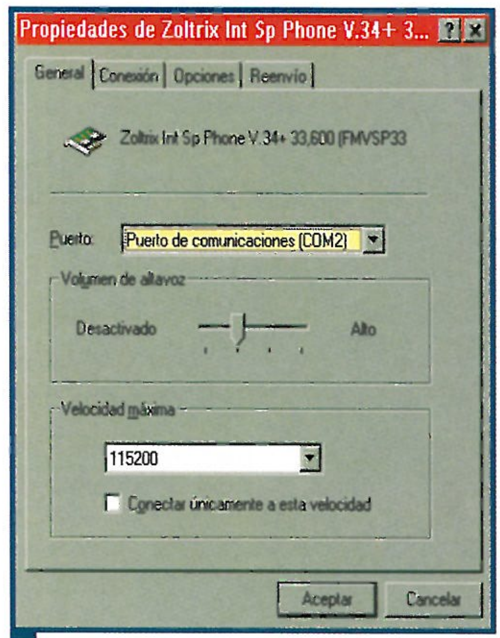


Figura 6.- Propiedades del *módem* instalado. Estas propiedades las tendrá en consideración *Windows.*

¿No conoces aún nuestras revistas?

Ahora lo tienes muy fácil



Si estas interesado en alguna de ellas te enviaremos un ejemplar gratuitamente

Sólo tienes que escribirnos una carta a la siguiente dirección:
C/ San Sotero nº 5, 1ª plta. Madrid 28037



DNS a fondo (y IV): Avanzado

Vicente Antonio Sánchez Werner.
Desarrollador independiente.

Después de implementar un servidor *DNS* en nuestra máquina *Linux,* en este artículo veremos la relación del *DNS* con el sistema *CIDR* de asignación de direcciones *IP,* las medidas de seguridad que podemos usar y un sistema para realizar modificaciones en las zonas de nuestro servidor de forma dinámica.

INTRODUCCIÓN

A diferencia de los demás artículos de la serie, en los que tratábamos un único tema, la estructura de éste es diferente. Hay muchos temas por tratar acerca de *DNS* avanzado, de los que sólo podemos tocar aquí una pequeña parte. Esa es la razón por la que este artículo está desglosado en pequeños capítulos en los que abordamos tres importantes temas, como son: la influencia del esquema de asignación de direcciones *IP CIDR,* la seguridad en un servidor *DNS* y la actualización dinámica de las zonas de *DNS.*



CLASES, CIDR Y DNS

En los inicios de *Internet* se fijó la forma en que las direcciones *IP* eran asignadas a las diversas sub-redes que la integraban. Este sistema se basaba en la existencia de unas asignaciones tipo, llamadas clases, que estaban compuestas por un número fijo de direcciones *IP* y que eran asignadas a las organizaciones según un criterio de tamaño.

Imagen 1.- Imagen de la página para registrarse en alguna de las listas de correo de *BIND.*

Inicialmente existieron tres clases llamadas A, B y C, que asignaban bloques de direcciones conteniendo 16 millones, 64000 y 256 direcciones *IP.* Este sistema implicaba que si una organización necesitaba sólo 16 direcciones para su red, recibía una clase C que desaprovechaba 240; si esta misma organización requería más de 256 direcciones podía obtener varias clases C o una clase B. En cualquier caso se estaba desaprovechando el espacio de direcciones, y aunque existían más de 4 mil millones de direcciones posibles esta situación provocó que si se seguía con este sistema de asignación de

direcciones *Internet* agotaría todas las posibles entre 1994 y 1995.

Acompañando el desmesurado crecimiento de la Red, desde el año 1992 se disparó el tamaño de las listas de encaminamiento mucho más de lo realmente necesario por el gran número de redes que tenían clases que no aprovechaban enteramente sus posibilidades.

En este punto, la *IETF* decidió usar otro sistema de asignación de direcciones, que resolviese el problema de su agotamiento, a la vez que optimizase de alguna forma el sistema de encaminamiento y permitiese un crecimiento más moderado de las citadas listas. Este sistema fue el *CIDR*.

El sistema *CIDR* fue la respuesta a estos problemas mientras ya se trabajaba en la nueva versión de los protocolos de *Internet*, en los que ya no habría el problema de direcciones al utilizar espacios tremendamente grandes (128 *bits* contra 32 *bits*). Éste proponía la sustitución de las clases por un sistema llamado de prefijos en el que la máscara de red indicaba el número de *bits* significativos de la red, prefijo o parte fija de la misma.

TABLA 1. *CIDR*/Clases.

Valor de la máscara (hexadecimal)	Valor de la máscara(decimal)	CIDR (Bits de prefijo)	Número de direcciones	Número de redes de clase (A, B o C) a las que equivale
80.00.00.00	128.0.0.0	/1	2048 M	128 A
C0.00.00.00	192.0.0.0	/2	1024 M	64 A
E0.00.00.00	224.0.0.0	/3	512 M	32 A
F0.00.00.00	240.0.0.0	/4	256 M	16 A
F8.00.00.00	248.0.0.0	/5	128 M	8 A
FC.00.00.00	252.0.0.0	/6	64 M	4 A
FE.00.00.00	254.0.0.0	/7	32 M	2 A
FF.00.00.00	255.0.0.0	/8	16 M	1 A
FF.80.00.00	255.128.0.0	/9	8 M	128 B
FF.C0.00.00	255.192.0.0	/10	4 M	64 B
FF.E0.00.00	255.224.0.0	/11	2 M	32 B
FF.F0.00.00	255.240.0.0	/12	1024 K	16 B
FF.F8.00.00	255.248.0.0	/13	512 K	8 B
FF.FC.00.00	255.252.0.0	/14	256 K	4 B
FF.FE.00.00	255.254.0.0	/15	128 K	2 B
FF.FF.00.00	255.255.0.0	/16	64 K	1 B
FF.FF.80.00	255.255.128.0	/17	32 K	128 C
FF.FF.C0.00	255.255.192.0	/18	16 K	64 C
FF.FF.E0.00	255.255.224.0	/19	8 K	32 C
FF.FF.F0.00	255.255.240.0	/20	4 K	16 C
FF.FF.F8.00	255.255.248.0	/21	2 K	8 C
FF.FF.FC.00	255.255.252.0	/22	1 K	4 C
FF.FF.FE.00	255.255.254.0	/23	512	2 C
FF.FF.FF.00	255.255.255.0	/24	256	1 C
FF.FF.FF.80	255.255.255.128	/25	128	1/2 C
FF.FF.FF.C0	255.255.255.192	/26	64	1/4 C
FF.FF.FF.E0	255.255.255.224	/27	32	1/8 C
FF.FF.FF.F0	255.255.255.240	/28	16	1/16 C
FF.FF.FF.F8	255.255.255.248	/29	8	1/32 C
FF.FF.FF.FC	255.255.255.252	/30	4	1/64 C
FF.FF.FF.FE	255.255.255.254	/31	2	1/128 C
FF.FF.FF.FF	255.255.255.255	/32	1	1/256 C

Fuente: RFC 1878

TABLA 2. *ACL* predefinidas

Nombre	Equivalencia
None	Ninguna dirección.
Any	Cualquier dirección.
Localhost	Todas las direcciones IP del servidor.
Localnets	Todas las direcciones IP de las redes en las que el servidor posea una interfaz de red.

Esta sustitución no requería un cambio radical en la organización y funcionamiento de la red y tampoco requería una migración forzosa de todas las redes al mismo, guardando compatibilidad con las antiguas clases a costa de que, en ocasiones, no se utilizasen las rutas más optimas para llegar a ellas.

CIDR incorporaba como novedad la posibilidad de crear sub-redes con un único integrante, o conceder direcciones *IP* en bloques mayores que las antiguas clases A en caso necesario, como se ve en la Tabla 1.

Algunos se preguntarán qué tiene que ver todo esto con el *DNS*. La respuesta es sencilla: el *DNS* depende de la asignación de las direcciones *IP* y de cómo se realiza en varios puntos; además, afecta de forma fundamental al funcionamiento del dominio *in-addr.arpa*.

En el artículo anterior supusimos que ambas redes eran de clase C, pero podría darse el caso de que no fuese así, teniendo 8 máquinas en una sub-red y 4 en la otra, desperdiciando dos clases C. Dada la naturaleza de nuestra red, podríamos optar por pedir un bloque de 16 direcciones *IP* para la primera y de 8 para la segunda red, ya que en ésta van a estar siempre cuatro máquinas (necesitamos una dirección más para realizar el *broadcast*).

En este caso pediríamos a nuestro proveedor de *Internet*, o a la entidad que nos proporciona las direcciones, una red *205.234.24/28*, indicando que 28 *bits* son significativos de la red y que vamos a usar 4 (el resto) para identificar a nuestras máquinas. La sub-red *204.234.209.0* se expresaría como *204.234.209/29*, usando 30 *bits* para marcar nuestra red y 3 para direccionar las máquinas (8 direcciones). Como podemos ver, esto implica un cambio en la máscara de red, y dado que nos están delegando partes del espacio de direcciones más pequeñas que las antiguas clases, también cambia la identificación de red (en este caso concreto- no, ya que tenemos el inicio de nuestras redes en *204.239.24.0* y en *204.239.209.0*), la dirección de *broadcast* y aparece un problema con el dominio *in-addr.arpa*.

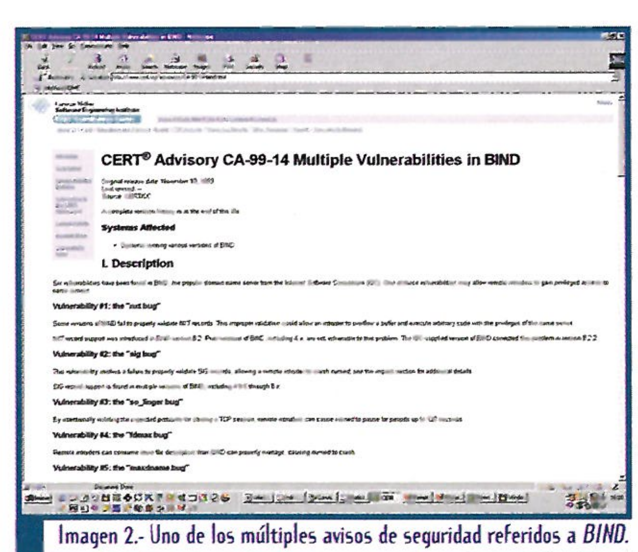


Imagen 2.- Uno de los múltiples avisos de seguridad referidos a *BIND*.

El problema aparece en la forma en que vamos a diferenciar las subredes que existen en una clase C. Si recordamos el artículo anterior, vemos que tendríamos tantos dominios *in-addr.arpa* como subredes, pero en este caso habría partes de ese espacio de direcciones que no nos corresponderían a nosotros, sino a otra red. Esto se soluciona recurriendo a nuestro proveedor de direcciones (ya que no existe un procedimiento uniforme para subdelegar dominios *in-addr.arpa* y cada entidad lo realiza de la forma que más le interese), para que nos informe exactamente cómo hacer el proceso o si ellos se encargan de todo.

SEGURIDAD EN SERVIDORES DNS

Un punto que puede preocupar a los administradores de red es el cómo implementar la seguri-dad en un servicio de *DNS* ya que puede abrir puertas y/o facilitar la elección de blancos en nuestra red por parte de un posible intruso.

Esta preocupación fue una de las motivaciones para el abandono de la versión 4.9.x de *BIND* y el desarrollo de la 8.x. En la 4.9.x apenas existían mecanismos de seguridad para restringir las peticiones y los accesos directos al servidor *DNS*, y se sabía que el propio *software* era sensible a algunos ataques que



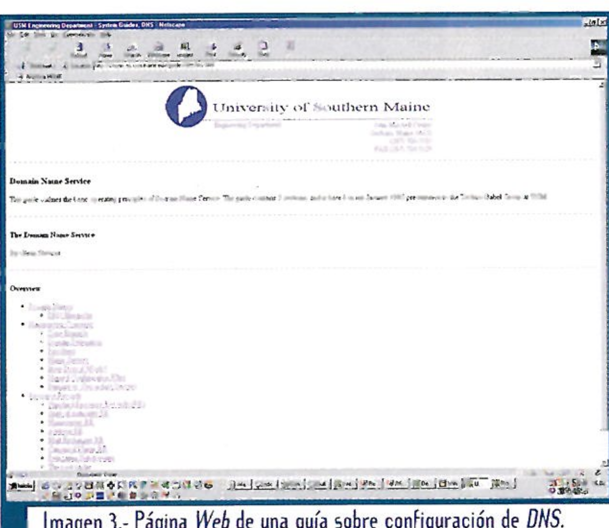


Imagen 3.- Página *Web* de una guía sobre configuración de *DNS*.

podían transformarse en importantes problemas de seguridad. *BIND* en su versión 8 vino a intentar solucionar la papeleta incorporando más mecanismos de seguridad y se reescribió el código para eliminar en la medida de lo posible las vulnerabilidades conocidas, problema aún no resuelto (prueba de ello es la aparición en pocas semanas de dos versiones parcheadas del *software*) y uno de los motivos por los que ya se trabaja en la versión 9 de *BIND*. En el momento de escribir este artículo, aparecía una nueva versión de *BIND*, la 8.2.2PL5 que añadía algunas mejoras y solucionaba la sensibilidad del programa a algunos ataques.

DNS dispone de un mecanismo para limitar el acceso a los servidores, y que es la base de todas las prestaciones de seguridad que podemos implementar en el servidor. Este mecanismo son las listas de control de acceso (*ACL*) que implementan una restricción de acceso mediante la agrupación de diversas *IP* y redes en diferentes grupos que poseerán diferentes prestaciones de seguridad.

Inicialmente existen por defecto 4 *ACL* predefinidas que se pueden ver en la Tabla 2, a las que podemos añadir las que necesitemos mediante la siguiente estructura en el archivo de configuración de *BIND*:

```
acl "nombre" {
  { lista de
direcciones };
};
```

Con esta sintaxis definimos una lista de control de acceso llamada **nombre** en la que se incluyen todas las direcciones incluidas en la lista o aquéllas que coincidan con el patrón que hayamos introducido. En la lista de direcciones podemos especificar direcciones y redes de estas dos formas:

- *dirección.de.red/bits*. Así indicamos una red, según la forma comentada en el apartado anterior, colocando la dirección de la red y los *bits* significativos (aquéllos que no varían). Así podemos introducir las redes que queremos que integren esta *ACL*.
- *dirección.de.red*. Así podemos introducir una dirección de red correspondiente a un único ordenador o servidor.

Ejemplos de estas sintaxis pueden verse en la Tabla 3.

Mediante este procedimiento podemos definir diferentes grupos que nos servirán para establecer nuestro esquema de seguridad aprovechando las nuevas prestaciones de *BIND* 8.x.

En un primer lugar hemos de preguntarnos si cualquier persona ha de poder mirar directamente nuestras zonas, accediendo a nuestro servidor, lo que supondría un acceso completo a nuestras zonas y sus datos. Obviamente en la mayoría de los casos esto no es deseable y lo normal es que deseemos restringir el acceso directo a máquinas de nuestra red, y a los servidores de nombres que nos interese. Esto se realiza mediante la estructura *allow-query* {*ACL* o lista de direcciones} que podemos incluir a nivel global o a nivel de cada una de las zonas, siempre teniendo en cuenta que la restricción por zonas tiene preferencia sobre la global. Un ejemplo de ambos casos puede ser:

TABLA 3. Ejemplos de creación de *ACL*

Ejemplo-1	acl "wind.com" {	{ 202.23.34.32/32;}; };
Ejemplo-2	acl "sulaco" {	{ 202.23.34.33;}; };
Ejemplo-3	acl "soloplinux.com" {	{ 202.23.34.0/29; 134.76.89.12;}; };
Ejemplo-4	acl "wind.com" {	{ 202.23.34.33; 134.76.89.12;}; };

```
options {
   allow-query { 209.234.24/24;
   130.11.22.1;
};

zone "soloprogramadores.com" {
    type master;
    file "db.solop";
    allow-query { 209.234.24/24;
          "wind.com"; };
};
```

En el primer caso decidimos que sólo contestaremos a las peticiones provenientes de la red *209.234.24.0* y de la máquina *130.11.22.1*. En la zona especificamos que se conteste a las peticiones provenientes de la red *209.234.24.0* y a las que proceden de las direcciones incluídas en la *ACL* con el nombre *"wind.com"*. Con estas simples modificaciones escondemos parte de nuestro espacio de nombres al resto del mundo, limitando el conocimiento previo que pueda tener un posible atacante sobre nuestra red.

Éste no es el único mecanismo de seguridad que podemos usar en nuestro servidor de nombres, ya que también podemos limitar otros aspectos para reducir aún más el acceso a la información.

La siguiente posibilidad aún es más poderosa, ya que permite limitar qué servidores de nombres pueden obtener la información completa de zona, transfiriéndola desde el servidor primario. Pensemos en el peligro que entraña el permitir un acceso completo a la información de zona, el posible atacante no sólo va a tener acceso al listado completo de las máquinas registradas que operan en nuestra red, sino que puede obtener información acerca de su configuración (si hemos usado comentarios o registros *hinfo*) y también acerca de la topología de la red y las máquinas que son susceptibles de ataque (por ejemplo aquéllas que tengan un registro *MX* son posibles candidatas ya que conocemos que al menos hay un servicio activo en ellas).

Por ello es fundamental que apliquemos restricciones a esta prestación para asegurarnos que sólo aquellas máquinas que van a hacer un uso legítimo (servidores esclavos) de la información contenida en las zonas puedan realizar la transferencia. Así mismo debemos vigilar el no caer en un error muy común, usar esta limitación sólo en los servidores primarios y no poner en los secundarios una limitación absoluta, lo que provocaría que un intruso tuviese acceso a la información completa de las zonas mediante la transferencia de las mismas desde los servidores esclavos que no impongan restricciones.

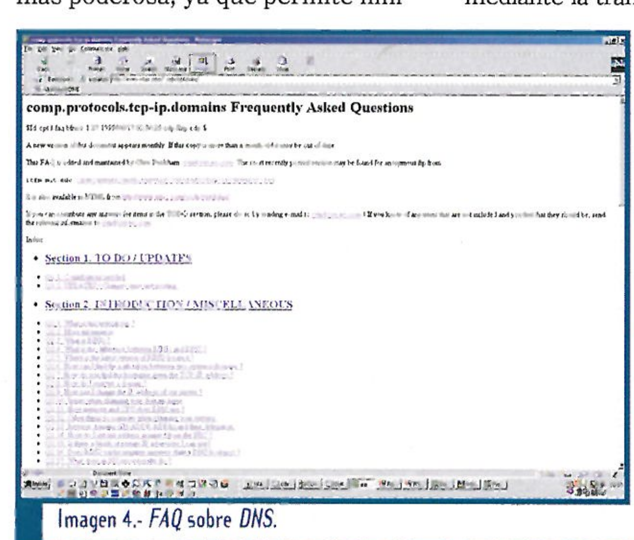


Imagen 4.- *FAQ* sobre *DNS*.

Esta característica se implementa mediante el uso de la orden *allow-transfer* {lista de direcciones} y puede ser implementada tanto a nivel global, como a nivel de zonas, aunque esta última restricción siempre tiene preferencia sobre la global. Un ejemplo de uso en una configuración de un servidor primario sería:

Para el servidor primario o maestro del dominio.

```
options    {
allow-transfer { 192.123.12.10;
  192.123.13.50;};
};

zone "ejemplo.com" {
type master;
file "db.ejemplo";
allow-transfer {192.123.12.10;
      193.123.12.35;};
};
```

Para no dejar ningún resquicio, en los servidores secundarios o esclavos deberíamos eliminar la orden de transferencia de la configuración de las zonas y denegar a todas las direcciones *IP* la posibilidad de realizar una transferencia de zona desde el servidor esclavo. Esto podría quedar así:

```
options    {
allow-transfer { none;};
};

zone "ejemplo.com" {
type slave;
masters {192.123.12.1;};
file "db.ejemplo";
};
```

De esta forma ninguna máquina podrá realizar una transferencia de zona desde el servidor esclavo, evitándonos posibles quebraderos de cabeza.

Finalmente, hemos de considerar la posibilidad de limitar en lo posible los privilegios a los que pueda tener acceso un posible intruso que se haya colado en nuestro servidor gracias a alguna vulnerabilidad del *software*, ya que si nuestro servidor se ejecuta a nivel de super-

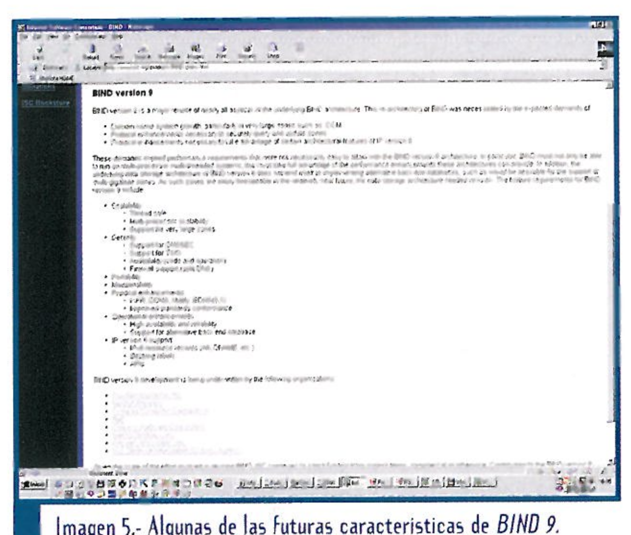


Imagen 5.- Algunas de las futuras características de *BIND 9*.

usuario, el intruso obtendrá los privilegios de superusuario pudiendo controlar totalmente el servidor.

Para evitar esta situación *BIND* introduce a partir de la versión 8 un código experimental que permite modificar el nivel de privilegios en el que se ejecuta el servidor, permitiendo así colocarlo con los mínimos privilegios posibles y limitando el acceso al sistema. Adicionalmente se ha incluido una opción que literalmente atrapa al servidor de nombres en su propio directorio. Así en caso de intrusión nuestro atacante se verá confinado a una pequeña porción del sistema de archivos. En ambos casos, estas opciones se especifican en la línea de comandos mediante los siguientes parámetros:

- *–u UID*. Indica qué usuario va a usar el servidor una vez haya arrancado.
- *–g GID*. Indica el grupo del usuario que va a usar el servidor tras su inicialización.
- *–t directorio*. Indica en qué directorio va a ser confinado el servidor tras su puesta en marcha.

Cuando usemos las dos primeras opciones hemos de tener cuidado de que los archivos de zonas y los archivos de las zonas actualizadas dinámicamente sean legibles para el servidor ya que en caso contrario podríamos encontrarnos con problemas. Asimismo en este caso existe la posibilidad de que, al cambiar de usuario, se sobreescriba el archivo *named.pid* (contiene la identificación del proceso) y falle la operación al no tener privilegios de acceso. La solución más común a esto es usar la opción *pid-file "directorio/named.pid"* dentro del bloque *options* del archivo de configuración, para colocar el archivo en un directorio sobre el que el servidor sí tenga privilegios de escritura. Finalmente hemos de procurar que si hubiésemos activado la opción de bitácora, ésta se escriba en un directorio donde tengamos permisos de escritura.

El uso de la opción *–t* es más complicado, ya que hemos de procurar que todos los archivos que el servidor vaya a usar se encuentren en el directorio al que lo vamos a confinar. Además de los archivos de configuración y de zonas, tenemos que colocar el ejecutable, el archivo *named-xfer*, las librerías dinámicas con que lo hayamos compilado (si corresponde) y finalmente un dispositivo */dev/null* que crearemos con la ayuda del comando */dev/MKDEV.* Mediante la utilización de estos parámetros crearemos un espacio propio para el servidor, que aislará al sistema de posibles intrusos que se introduzcan gracias a vulnerabilidades del servidor de nombres.

La aplicación de todos estos elementos de seguridad nos permite crear una seguridad, no sólo alrededor de nuestro servidor, sino también de nuestra red.

ACTUALIZACIÓN DINÁMICA DE SERVIDORES DNS

Antiguamente, cuando deseábamos realizar una actualización de los archivos de zona de un servidor de *DNS*, debíamos editarlos y sobreescribir los antiguos, cambiando el número de serie de forma manual. La introducción de *BIND* 8 incorporó la posibilidad de realizar este proceso sin necesidad de editar los archivos, permitiendo la modificación dinámica de sus contenidos en tiempo real.

Esta característica se usa en sistemas de *DNS* dinámico como *DHCP* o *Dynips*, y permite dar de alta y baja los registros del *DNS* según las máquinas se vayan conectando o desconectando de nuestro servicio. Aunque ésta es su aplicación más habitual, también es posible utilizar el programa *nsupdate* que viene en la distribución de *BIND* para realizar este proceso de forma manual.

Antes de proceder a utilizar esta característica hemos de modificar la configuración del servidor para que permita realizar las modificaciones dinámicamente. Al contrario que otras opciones comentadas en este artículo, *allow-update* no puede especificarse de forma global, sino que se ha de activar en cada una de las zonas que deseamos modificar dinámicamente. La sintaxis de esta opción es similar a las anteriores y toma como parámetro una *ACL* o una lista de direcciones *IP* desde las que vamos a permitir la actualiza-

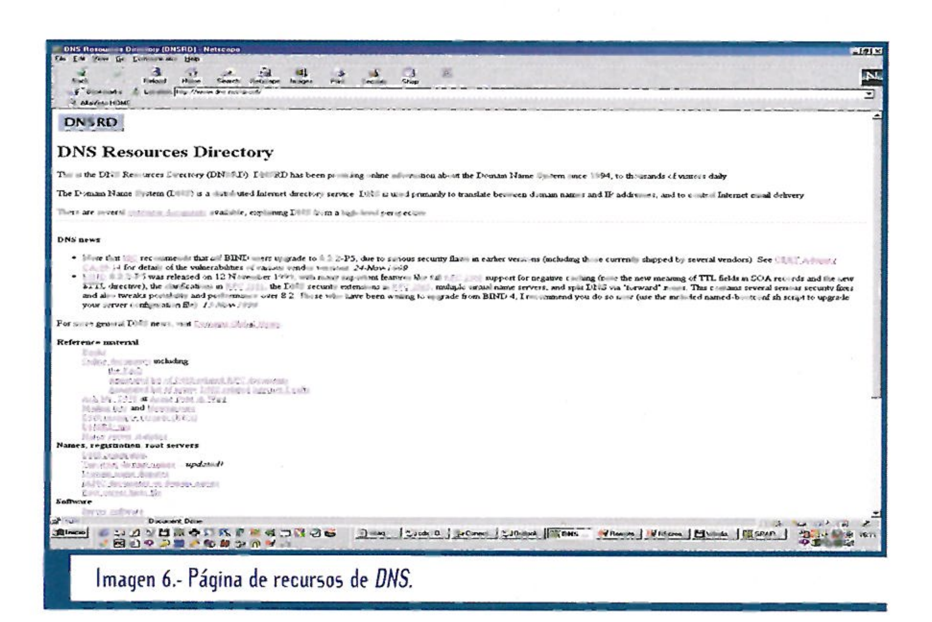


Imagen 6.- Página de recursos de *DNS*.

ción dinámica. A continuación se muestra un ejemplo:

```
zone "space.es" in {
type master;
file "zn.space";
allow-update { 209.234.24.6;};
};
```

Con esto le indicaríamos a nuestro servidor maestro que la zona **"space.es"** se puede modi-ficar de forma dinámica y que sólo debe admitir aquellas modificaciones provenientes de **209.234.24.6.** Como hemos visto, realizamos la modificación en nuestro servidor maestro y no en los esclavos, ya que aunque éstos sean servidores autorizados, no poseen capacidad de modificar los datos de las zonas originales y por lo tanto no tiene mucho sentido que realicemos las modificaciones en ellos, ya que en el siguiente refresco del archivo de zonas se perderán los cambios que hayamos realizado.

Una vez que hemos realizado esta modificación ya podemos comenzar a usar el servicio de actualización dinámica desde la máquina/s a las que hayamos dado permiso mediante la ejecución del comando *nsupdate*. Una vez ejecutado muestra una línea de comandos en la que iremos introduciendo los comandos para actualizar y mantener la zona. Los comandos que esta utilidad entiende son los siguientes:

- *prereq yxrrset ndominio tipo [datos]*. Este comando obliga a que, para realizar una modificación, deba existir un registro *tipo* perteneciente al dominio *ndominio* y que, en caso de especificar los datos, el contenido del registro coincida con los proporcionados al comando.

Nota: En cualquier comando de nsupdate, ndominio puede ser tanto un dominio o un FLQDN.

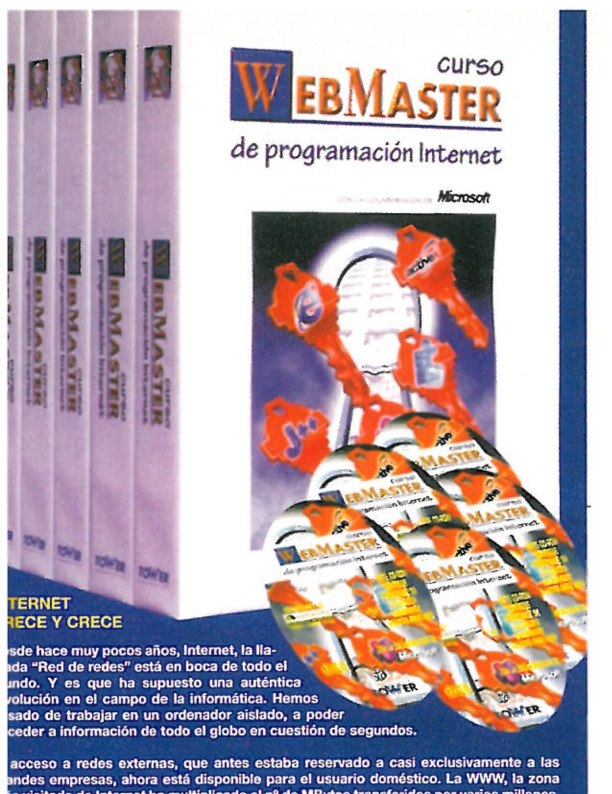
- *prereq nxrrest ndominio tipo [datos]*. Es el caso contrario, el requisito es la no existencia del registro especificado.
- *prereq yxdomain ndominio*. Obliga a que para realizar la actualización exista el dominio *ndominio*.
- *prereq nxdomain ndominio*. Es el caso opuesto, la no existencia del dominio *ndominio* es la condición para realizar la actualización.
- *update delete ndominio [tipo] [datos]*. Elimina el nombre del dominio dado o, si especificamos el tipo, borra el registro de ese tipo o si incluimos los parámetros del tipo borrará el registro que coincida exactamente con *ndominio*, *tipo* y *datos*.
- *update add ndominio ttl [clase] tipo datos*. Crea el registro de clase *clase* (en caso de no especificarla será *IN*), de tipo *tipo*, que almacenará los datos *datos* pertenecientes al dominio *ndominio*.

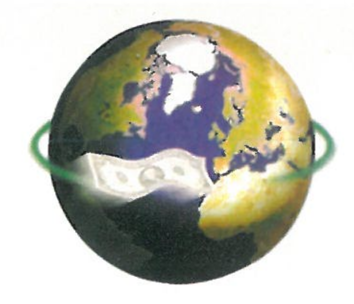
Con estos comandos ya podremos realizar las modificaciones dinámicas que más nos interesen y de forma segura, ya que sólo se podrán realizar desde el punto que nosotros hayamos indicado.

CONCLUSIÓN

Con estos tres temas acabamos esta serie de artículos sobre el protocolo *DNS*, uso e implementación en máquinas *Linux*. Como hemos podido ver esto es un mundo en sí mismo, que tras su aparente sencillez esconde una complejidad y unas posibilidades asombrosas que ahora podemos comenzar a ver.

En el tintero se han quedado temas muy interesantes que de haberlos tratado harían que esta serie se prolongase mucho más de lo deseable, debido a la complejidad que encierran; entre estos temas quedan los servicios de *DNS* dinámico, actualización mediante autentificación de usuarios y otros temas que abarcan aspectos más profundos, como el control de los cachés de los servidores o la depuración de los fallos en nuestros sistemas.

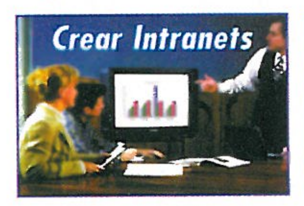




- ✓ En España se ha duplicado el número de usuarios en los últimos 8 meses.
  Fuente: AIMC
- ✓ El 80% de internautas existentes en todo el mundo entran en la Red para realizar negocios.
  Fuente: FORRESTER RESEARCH
- ✓ 100 millones de personas navegan hoy.
  Fuente: Varios compilado por NUA INTERNET SURVEYS
- ✓ En 5 años el comercio electrónico supondrá 4,5 billones de pesetas tan sólo en los Estados Unidos.
  Fuente: IBID
- ✓ En el año 2.000 la gran mayoría de las transaciones no realizadas en dinero metálico serán mediante pagos electrónicos.
  Fuente: KILLEN ASSOCIATES

PREPÁRATE PARA SER EL MEJOR WEBMASTER

Aprenderás a:













BOLETÍN DE SUSCRIPCIÓN

OFERTA VÁLIDA SÓLO PARA ESPAÑA

SÍ, deseo suscribirme al CURSO WEBMASTER DE PROGRAMACIÓN INTERNET

*** Oferta válida hasta fin de existencias** *** Entregas mensuales de 80 fichas**

CONDICIONES DE PAGO: (MARQUE LA OPCIÓN DESEADA)

❑ Pago al contado: A partir de la tercera entrega.
28.490 Ptas.
Se entregará **gratis** la versión de **FrontPage 98** en el primer envío. Valorado en 22.272 Ptas.

❑ Pago a plazos:
En 12 mensualidades de 2.995 Ptas.
En el cuarto envío se incluirá **gratis FrontPage98.**
Total: 35.940 Ptas.

UTILICE MAYÚSCULAS PARA RELLENAR ESTA TARJETA

Nombre y apellidos .... F. nacimiento......................

Domicilio .... C.P. ....

Ciudad .... Provincia .... Telf. ....

e-mail .... Profesión ....

FORMAS DE PAGO:

❑ Con **cargo a mi tarjeta VISA** Nº ____ ____ ____ ____ Caduca ....

❑ **Domiciliación bancaria**

CÓDIGO CUENTA CORRIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Sr. Dtor. del banco ....

Población ....

Ruego a UD. que se sirva cargar en mi ❑ cuenta corriente ❑ libreta de ahorro el/los recibos que le será presentado por REVISTAS PROFESIONALES, S.L. como pago de mi suscripción al **CURSO WEBMASTER DE PROGRAMACIÓN INTERNET**

❑ **Cheque** a nombre de REVISTAS PROFESIONALES, SL.

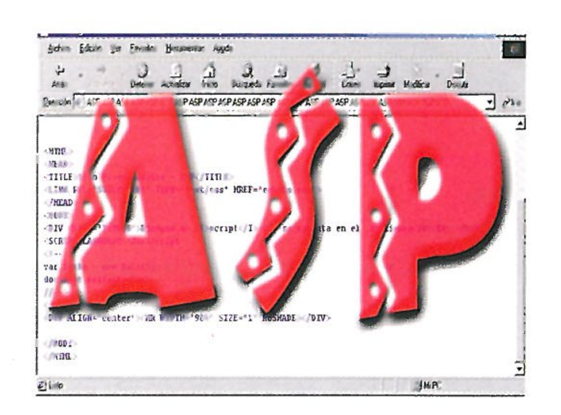
❑ **Contra reembolso** del importe más gastos de envío.

❑ **Giro Postal** (adjunto fotocopia del resguardo)*.

Firma

*Esta forma de pago es exclusivamente para pagos al contado





La tecnología ASP (IV): Componentes ActiveX de servidor y BBDD

Adolfo Aladro García. *Analista programador*

Los componentes *ActiveX* de servidor proporcionan una forma de hacer ciertas operaciones a partir de objetos predefinidos, lo que facilita enormemente la labor del desarrollador de aplicaciones *Web*. Asimismo estos componentes permiten acceder a todo tipo de fuentes de datos de forma rápida y sencilla.

COMPONENTES ACTIVEX DE SERVIDOR

Los componentes *ActiveX* de servidor forman parte de las aplicaciones *Web* y son invocados desde los *scripts* de servidor de las páginas *ASP*. De esta manera la funcionalidad que proporcionan queda encapsulada y los desarrolladores pueden contar siempre con ese recurso sin tener que preocuparse por su funcionamiento interno. Existen cinco componentes *ActiveX* de servidor que ofrece por defecto la plataforma *ASP*:

- **Ad Rotator**
  Este componente se utiliza para visualizar alternativamente una serie de imágenes, que normalmente se corresponderán con anuncios o *banners*. Podemos configurar el sistema de rotación de imágenes para que cada una de ellas enlace a una dirección *URL* concreta. La lista de imágenes que mostrar se almacena en un archivo de texto plano y el componente *Ad Rotator* las muestra dependiendo de la configuración.

- **Browser Capabilities**
  Este componente permite conocer las características del navegador del cliente que ha realizado una determinada petición al servidor. Esta información es útil cuando las páginas que han de servirse dependen de las características del navegador.

- **File Access**
  En el artículo anterior de esta serie vimos, por primera vez, este componente al realizar el ejemplo del administrador de colores. Permite acceder al sistema de ficheros del servidor y realizar todo tipo de operaciones tales como leer y escribir ficheros, modificar la estructura de directorios y subdirectorios, etc.

- **Content Linking**
  El componente de enlace a contenidos gestiona una lista de direcciones *URL* de forma que partiendo de nuestro sitio *Web* podemos hacer



tablas de contenido o sistemas de navegación.

- **Database Access**
  El componente de acceso a base de datos proporciona la posibilidad de incorporar bases de datos a nuestras aplicaciones *Web*. Las consultas a las bases de datos se producen en el servidor y los resultados devueltos pueden ser manipulados antes de integrarse con el resto de la página que finalmente verá el cliente.

Los componentes facilitan la labor del desarrollador

EL COMPONENTE AD ROTATOR

Uno de los elementos que aparecen con más frecuencia en los sitios *Web* son los denominados *banners*. Estos responden a un sistema de publicación de anuncios que indica qué imágenes han de mostrarse cada vez que un cliente solicita una página *Web*. Estas imágenes normalmente suelen contener publicidad y en muchos casos éstas son a su vez enlaces que conducen al sitio *Web* del anunciante. El componente *Ad Rotator* sirve para gestionar todos estos aspectos de una forma cómoda y práctica.

Figura 1.- Ejemplo de utilización del componente *Ad Rotator*.

Ad Rotator sirve para automatizar sistemas que contienen banners

Para implementar un sistema de anuncios basta con escribir las dos líneas que siguen en las páginas *ASP* que van a contener los *banners* con publicidad.

```
<% var ad =
     Server.CreateObject("MSWC.AdR
     otator") %>
<%= ad.GetAdvertisement("adrota-
     tor.txt") %>
```

La primera línea crea una instancia del componente, mientras que la siguiente hace que el objeto anteriormente instanciado lea la información de configuración de un archivo de texto plano. Este archivo es el que dicta al componente cómo debe comportarse. En otras palabras: qué imágenes conforman el sistema de anuncios, cuáles son sus características (anchura, altura y borde), qué importancia relativa tienen las imágenes dentro del sistema, cuáles son las direcciones *URL* a las que apuntan y si existe una página *ASP* encargada de realizar la redirección.

SINTÁXIS DEL FICHERO DE CONFIGURACIÓN

El componente *Ad Rotator* toma los parámetros que determinan su comportamiento de un fichero de configuración. Este fichero de texto plano tiene una sintaxis mediante la cual podemos indicar todas las características de un componente en concreto. Existen dos secciones separadas por un asterisco (*).

```
[REDIRECT URL]
[WIDTH numWidth]
[HEIGHT numHeight]
[BORDER numBorder]
*
adURL
adHomePageURL
Text
impressions
```

La primera de las líneas indica la dirección *URL* de la página *ASP* encargada de realizar la redirección. Este parámetro es opcional y suele utilizarse en aquellos casos en los que es interesante controlar el tráfico de usuarios. Esto se produce en casi todos los sistemas de *banners* que existen en *Internet*. Los anunciantes pueden utilizar esa página para almacenar datos concretos de la redirección y comprobar así la efectividad de los anuncios en cada uno de los sitios *Web* donde aparecen.

A continuación se indican la anchura y altura de la imagen así como si se desea que ésta aparezca con borde o no. Estos parámetros se traducirán a los correspondientes atributos de la etiqueta *IMG* (*WIDTH*, *HEIGHT* y *BORDER*) en el código *HTML* que llega al navegador del cliente.

Impressions determina la importancia relativa de un anuncio

La siguiente sección sirve para especificar las características particulares de cada una de las imágenes que se muestran. La línea *adURL* contiene la dirección *URL* de la imagen que aparecerá en la página. La siguiente línea, *adHome*

LISTADO 1. Fichero configuración del componente *Ad Rotator*.

```
>---ADROT.TXT---
REDIRECT http://127.0.0.1/asp/redireccionar.asp
WIDTH 460
HEIGHT 60
BORDER 1
*
http://127.0.0.1/asp/banner01.gif
-
Este es el banner 01
2
http://127.0.0.1/asp/banner02.gif
-
Este es el banner 02
2
http://127.0.0.1/asp/banner03.gif
-
Este es el banner 01
3
http://127.0.0.1/asp/banner04.gif
-
Este es el banner 04
1
```

PageURL, guarda la dirección *URL* de la página principal del sitio *Web* vinculado con esa imagen. En el caso de que no exista, se escribe en su lugar un guión -. La línea *Text* almacenará la cadena de texto que aparece en el atributo *ALT* de la etiqueta *IMG*. Éste se mostrará en aquellos casos en los que el navegador no sea capaz de visualizar las imágenes.

La cabecera HTTP User Agent revela el navegador del cliente

Finalmente la última de las líneas, *impressions*, contiene un número que puede oscilar entre **0** y **4294967295.** Este valor sirve para determinar la importancia que tienen las imágenes. Así si tenemos un componente formado por cuatro imágenes y sus valores respectivos son 4, 3, 1 y 2, significa que el 40% del tiempo aparecerá la imagen primera, el 30% la segunda, el 10% la tercera y el 20% la última. El Listado 1 muestra un ejemplo de fichero de configuración del componente *Ad Rotator*.

EL COMPONENTE BROWSER CAPABILITIES

Este componente sirve para determinar las características del navegador. Dado que todos los navegadores tienen sus particularidades, en muchas ocasiones es preciso distinguir entre unos y otros a la hora de servir las páginas de nuestros sitios *Web*. De esta manera podemos crear aplicaciones en las que todos los clientes vean aproximadamente lo mismo con independencia del navegador que utilicen.

El funcionamiento de este componente es muy sencillo. Cuando un navegador se conecta con un servidor *Web* automáticamente manda una cabecera *HTTP* denominada *User Agent*. Esta cabecera es una cadena de texto que identifica al navegador. El componente *Browser Capabilities* compara esta cabecera con las entradas que aparecen en el archivo *Browscap.ini*. Si el resultado de dicha comparación es positivo, el componente asume las propiedades correspondientes. En otro caso toma las propiedades por defecto. El archivo *Browscap.ini* se encontrará en un sitio distinto dependiendo de la instalación del servidor. En el caso concreto de la instalación descrita en el primer artículo de esta serie, su ubicación es *C:\WINDOWS \SYSTEM\inetsrv.*

Para crear una instancia del componente lo primero que debemos hacer es:

```
<% var naveg =
    Server.CreateObject("MSWC.Bro
    wserType") %>
```

Posteriormente, y siempre dependiendo del contenido del archivo de especificación de los navegadores, podremos acceder a las distintas propiedades del navegador de la siguiente manera:

```
<%= naveg.browser %>
```

EL COMPONENTE CONTENT LINKING

Este componente sirve para gestionar enlaces. Esto significa que en alguna parte se almacenan una serie de enlaces con textos que los describen y el componente toma de ahí



su información para construir índices, sistemas de navegación dentro del sitio *Web* o tablas de contenidos.

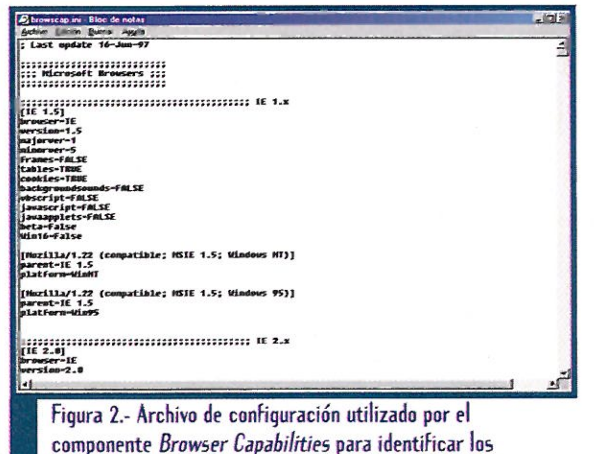
```
<% var nl = Server.CreateObject
    ("MSWC.NextLink") %>
    <% var n = nl.GetListCount
    ("nextlink.txt") %>
```

Las dos líneas anteriores sirven para instanciar e inicializar respectivamente el componente. Como se puede apreciar, éste toma la lista de direcciones *URL* de un fichero de texto, algo similar a lo que ocurre con la configuración de los componentes *Ad Rotator*.

Debemos configurar la base de datos como origen de datos en el Panel de Control de Windows

El fichero de configuración está formado por líneas de texto separadas entre sí por retornos de carro. Cada línea puede tener varios campos y éstos se separan a su vez con tabuladores. La sintaxis es la siguiente:

- **Enlace**
  Dirección *URL* virtual o relativa dada en el formato *nombre_fichero* o *directorio/nombre_fichero*. Las direcciones *URL* abso-lutas (aquellas que comienza por *"http://"*, *"//"* o "\\") no están contem-pladas.



```
; Last update 16-Jun-97

;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Microsoft Browsers ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; IE 1.x
[IE 1.5]
browser=IE
version=1.5
majorver=1
minorver=5
frames=FALSE
tables=TRUE
cookies=TRUE
backgroundsounds=FALSE
vbscript=FALSE
javascript=FALSE
javaapplets=FALSE
beta=False
Win16=False

[Mozilla/1.22 (compatible; MSIE 1.5; Windows NT)]
parent=IE 1.5
platform=WinNT

[Mozilla/1.22 (compatible; MSIE 1.5; Windows 95)]
parent=IE 1.5
platform=Win95

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; IE 2.x
[IE 2.0]
browser=IE
version=2.0
```

Figura 2.- Archivo de configuración utilizado por el componente *Browser Capabilities* para identificar los navegadores.

- **Descripción**
  Cadena de texto que describe el enlace.

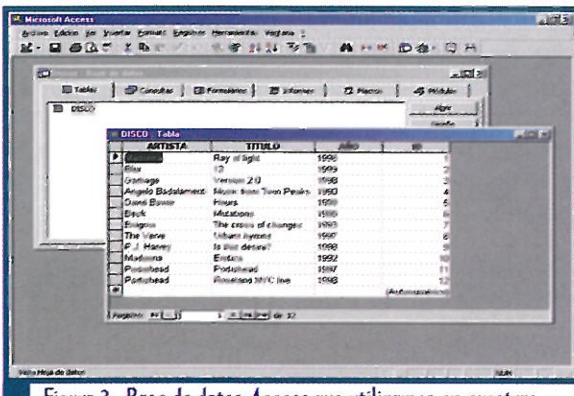


Figura 3.- Base de datos *Access* que utilizamos en nuestras páginas *ASP*.

- **Comentario**
  Cadena de texto que funciona como comentario, es decir, que no será procesada por el componente al ser cargado.

Un típico ejemplo en el que el uso de este componente simplifica mucho la labor de los desarrolladores lo encontramos al crear un sistema de navegación típico en el que cada página enlace a la siguiente y a la anterior.

```
<% if (nl.GetListIndex("nex-
    tlink.txt") > 1) { %>
<a href="<%=
    nl.GetPreviousURL("nextlink.t
    xt")%>">
Pág. Anterior</a>
    <% } %>
    <a href="<%=
    nl.GetNextURL("nextlink.txt")
            %>">
            Pág. Siguiente</a>
```

El método *etListIndex* devuelve un valor que se corresponde con la posición que ocupa la dirección *URL* actual dentro del fichero de configuración. Los métodos *GetPrevious URL* y *Get NextURL* devuelven respectivamente las direcciones anterior y siguiente dentro del archivo de configuración.

ACCESO A BASES DE DATOS

Probablemente uno de los componentes *ActiveX* de servidor más utilizados sea el que permite acceder a bases de datos. Se trata de un tema extenso que aquí abordaremos desde una perspectiva práctica y sin complejidades, por lo que recomendamos a todos aquellos que estén interesados que lean toda la abundante documentación proporcionada por *Microsoft*. Ésta se puede instalar junto con el propio servidor y se muestra vía *Web*. Para la instalación propuesta en el primero de los capítulos de esta serie, podemos encontrarla en la dirección *URL* **http://localhost/iisHelp/pws/misc/default.asp.**

Hoy en día prácticamente no existen sitios *Web* de cierta envergadura que no tengan acceso a bases de datos. Esto plantea una problemática ciertamente difícil de resolver que abarca desde las particularidades de las plataformas donde se almacenan los datos hasta los mecanismos concretos mediante los cuales se extrae y actualiza la información.

A continuación se propondrá un modelo sencillo basado en la base de datos *Access*. La forma en la que se accede a los datos será

equivalente para cualquier otra configuración. El componente *ActiveX* de servidor *Database Access* utiliza *ADO* (*ActiveX Data Objects*) para poder acceder a los datos almacenados en una base de datos.

ADO (ACTIVEX DATA OBJECTS)

Tras las siglas *ADO* se encuentra una tecnología que permite acceder a bases de datos para consultar y manipular datos. *ADO* proporciona una serie de objetos mediante los cuales la interacción con fuentes de datos se lleva a cabo de forma sencilla y rápida.

- **Command**
  Los objetos de tipo *Command* sirven para guardar información acerca de los comandos que se pueden lanzar contra una conexión o incluso contra los resultados que devuelve una consulta. Se utiliza principalmente en consultas con parámetros o procedimientos almacenados que devuelven resultados a través de parámetros.

- **Connection**
  Los objetos de tipo *Connection* representan las conexiones que mantiene el servidor con el origen de datos.

- **Error**
  Los objetos de tipo *Error* proporcionan información detallada de los errores que se pueden producir en el acceso a las bases de datos.

- **Field**
  Cuando se realiza una consulta a la base de datos normalmente ésta suele devolver una tabla de resultados con una o más columnas. Los objetos de tipo *Field* representan las columnas devueltas.

- **Parameter**
  Los objetos de tipo *Parameter* sirven para guar-dar información acerca de los parámetros utilizados en las diversas operaciones relacionadas con el acceso a la base de datos.

- **Property**
  Los objetos de tipo *Property* representan propiedades o características de los objetos *ADO*.

- **Recordset**
  Los objetos de tipo *Recordset* representan las filas que componen el resultado devuelto por una consulta a la base de datos.

El componente ActiveX de servidor Database Access utiliza ADO (ActiveX Data Objects)

Además de los objetos anteriores, que cuentan con numerosos métodos y propiedades, existen también otros objetos que se definen como colecciones. Así por ejemplo todo objeto de tipo *Recordset* contiene a su vez una colección de objetos de tipo *Field* llamada *Fields* que, como es evidente, guarda una relación de cada una en las columnas resultantes de la consulta a la base de datos.

ARCHIVO DE DEFINICIÓN DE LAS CONSTANTES ADO

ADO proporciona una serie de constantes que se utilizarán con frecuencia en las páginas *ASP* que acceden a bases de datos. Estas constantes están recogidas en un archivo que debe ser incluido en la página *ASP* antes de realizar ninguna operación. A este proceso se le denomina inclusión de servidor (*SSI*) y para ello las páginas *ASP* cuentan con la directiva *include*.

Dependiendo del lenguaje de *script* (*VBScript o JScript*) utilizado tendremos que incluir un archivo diferente.

```
<!--#include virtual="ADOVBS.INC"-
    ->
<!--#include
     virtual="ADOJAVAS.INC"-->
```

Estos archivos se encuentran en el servidor cada vez que éste se ha instalado. Normalmente estarán en el directorio *c:\Archivos de programa\Archivos comunes\SYSTEM \ADO\Docs.*

UN EJEMPLO

Lo primero que debemos hacer es configurar la base de datos como origen de datos. Para ello debemos ir al *Panel de Control* de *Windows* y hacer *clic* sobre el icono Fuentes de datos *ODBC*. Entonces seleccionamos la pestaña **DNS** de sistema y hacemos *clic* en el botón **Agregar**. El controlador elegido será *Microsoft Access Driver (*.mdb)* y en la ventana de configu-

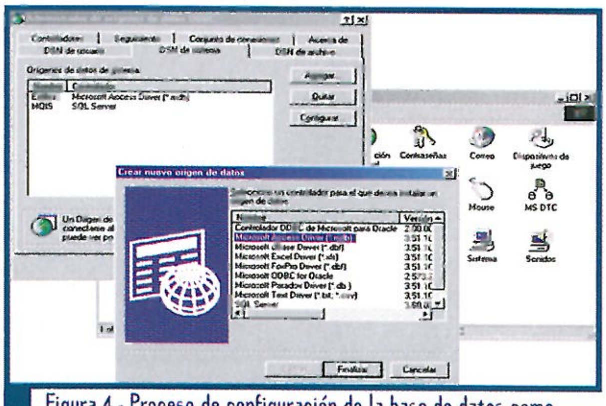


Figura 4.- Proceso de configuración de la base de datos como fuente de datos.



ración será necesario especificar la ruta al archivo con extensión *MDB* que contiene la base de datos así como el nombre con el que se va a conocer. Una vez que hemos hecho esto, nuestra base de datos está lista para ser usada desde las páginas *ASP*.

Para el ejemplo que nos ocupa se ha desarrollado una pequeña base de datos en la que solamente hay una tabla llamada *DISCO*. Esta tabla almacena los datos de una colección de discos y tiene tres columnas: *ARTISTA, TITULO, AÑO* e *ID*.

En la página *ASP* que va a mostrar el contenido de la base de datos lo primero que debemos hacer es instanciar un objeto de tipo *Connection* mediante el uso del método *CreateObject* del objeto *Server*.

```
var c =
    Server.CreateObject("ADODB.Co
    nnection");
```

En este momento la variable **c** contendrá una referencia al objeto de tipo *Connection* encargado de establecer la comunicación con el origen de datos. Para abrir la base de datos deseada tenemos que utilizar el método *Open* de la conexión. Este método puede recibir diferentes parámetros en función de la configuración de la base de datos como fuente de datos, pero para este ejemplo bastará con pasar como parámetro el nombre que le dimos en el *Panel de Control*.

```
c.Open("DISCO");
```

La sentencia anterior marca el inicio de una conexión a una base de datos concreta. Ahora podemos empezar a realizar consultas *SQL* y recoger los datos.

```
var rs = c.Execute("SELECT * FROM
    DISCO ORDER BY ARTISTA");
```

Ejecutar una sentencia *SQL* contra la base de datos es tan sencillo como hacer una llamada al método *Execute* en la que se pasa como parámetro una cadena de texto que contiene la consulta. Este método devuelve como resultado un objeto de tipo *Recordset*, el cual recoge los datos obtenidos como consecuencia de la consulta.

Una de las propiedades de un objeto de tipo *Recordset* es la colección de elementos denominada *Fields*. Este objeto contiene información sobre las columnas que constituyen el resultado devuelto por la consulta. Así podemos recorrer la colección *Fields* y extraer, por ejemplo, el nombre de dichas columnas. Para ello la propiedad *Count* nos informa acerca del número de las mismas.

```
<TR>
<% for (i = 0; i <
    rs.Fields.Count - 1; i++) {
    %>
    <TD><B><%= rs.Fields(i).Name
    %></B></TD>
<% } %>
</TR>
```

Para referirnos a un objeto de tipo *Field* en concreto, dentro de la colección *Fields*, utilizamos la sintáxis *rs.Fields(i)* donde *i* es el índice que indica la columna a la que nos referimos. La propiedad *Name* de un objeto de tipo *Field* proporciona el nombre de la columna. El proceso de recorrer los datos devueltos por la consulta es similar.

```
<% while (!rs.EOF) { %>
    <TR>
    <% for (i = 0; i <
    rs.Fields.Count - 1; i++) {
    %>
        <TD VALIGN=TOP><%= rs(i)
    %></TD>
    <% } %>
    </TR>
    <%
    rs.MoveNext()
}
rs.Close()
c.Close()
%>
```

La propiedad *EOF* de un objeto de tipo *Recordser* informa acerca de cuándo hemos terminado de recorrer todos los datos devueltos. El método *MoveNext* permite ver secuencialmente una a una las filas devueltas. Para cada una de las filas accederemos a los datos utilizando de nuevo la colección *Fields*.

Cuando se ha terminado la operación debemos liberar recursos. Para ello contamos con el método *Close*, que se aplica a los objetos de tipo *Recorset* y a los de tipo *Connection*.

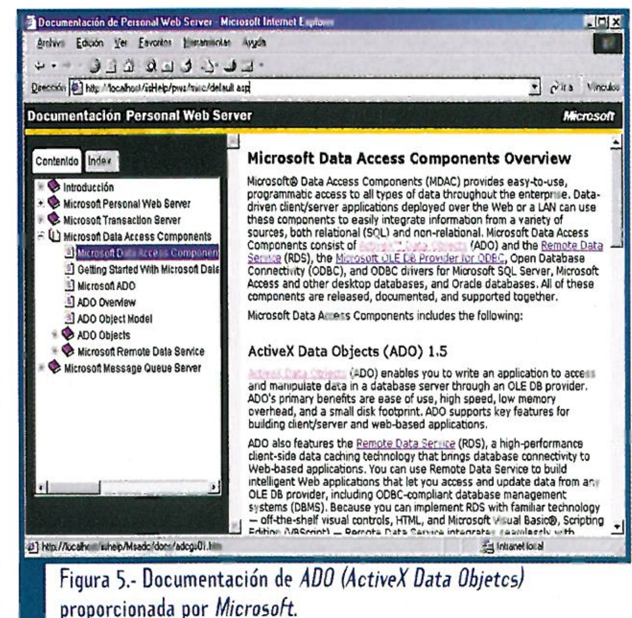


Figura 5.- Documentación de *ADO (ActiveX Data Objetcs)* proporcionada por *Microsoft*.

Criptografía (IV)

Javier Sanz Alamillo. *Ingeniero de Software.*

En este artículo se van a describir los algoritmos de clave pública, que en su momento revolucionaron la criptografía actual, y sus diferentes posibilidades de uso, como en intercambio de claves, firma digital, etc.

Si los algoritmos de clave privada mostrados en anteriores artículos son muy utilizados y con excelentes resultados, los algoritmos de clave pública no lo son menos y aunque tienen fama de ser más lentos y en cierta forma algo engorrosos por sus características propias, ofrecen un conjunto de ventajas frente a los anteriores que en muchas circunstancias los hacen casi imprescindibles.

ALGORISMOS DE CLAVE PÚBLICA

Los algoritmos de clave pública o asimétricos fueron introducidos por *Whitfield Diffie* y *Martin Hellman* a mediados de los años 70, siendo su principal novedad respecto a la criptografía simétrica que las claves no eran únicas, es decir, no se utilizaba la misma clave para el cifrado y el descifrado, sino que las claves se formaban por pares.

Si bien esto dió pie a la aparición de un gran conjunto de algoritmos asimétricos, muchos de ellos resultaron ineficaces al ser utilizados con grandes volúmenes de información o eran poco resistentes a ataques criptográficos.

Con el paso del tiempo surgieron algoritmos asimétricos seguros, basados en problemas matemáticos de muy difícil resolución, al menos en un tiempo razonable, y con claves de una longitud más que considerable, como **1024** *bits* o superiores.

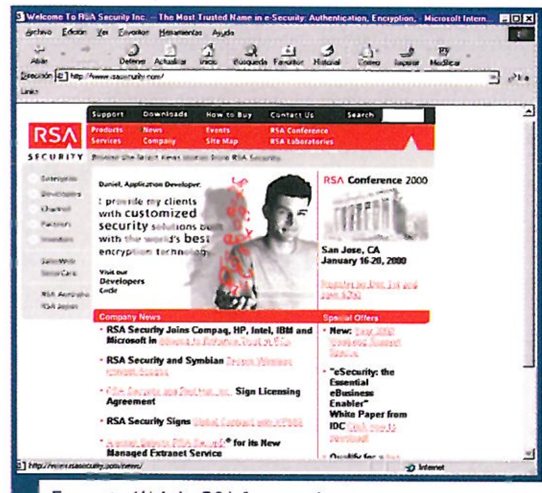


Figura 1.- Web de *RSA Security Inc.*

CARACTERÍSTICAS

Los criptosistemas de clave pública se caracterizan por utilizar dos claves para las operaciones de cifrado y descifrado. Una de ella, denominada clave pública se utiliza para el cifrado y no tiene por qué guardarse en secreto, y la otra, la clave privada, se aplica en el proceso de descifrado. Se trata de una clave secreta, que no debe conocerla nadie más que la persona que la utiliza.

En realidad, pueden intercambiarse los papeles de las claves y el resultado sería el mismo, pero por costumbre se suelen denominar de esta forma.

Este proceso de utilización de pares de claves se basa en la aplicación del concepto de función unidi-



LISTADO 1. Creación claves RSA.

```
public void createRSAKeys() throws Exception
{
    KeyPair kp;
    CipherKey dk = null ;

    FileOutputStream fd_pub = new FileOutputStream ("rsapub");
    FileOutputStream fd_pri = new FileOutputStream ("rsapri");

    kp = RSAKey.createKeys(512);

   System.out.println ( "RSA Publica : \n" + kp.getPublic() );
   System.out.println ( "RSA Privada : \n" + kp.getPrivate());

   byte [] tb_rsapub = new String (""+kp.getPublic()).getBytes();
   byte [] tb_rsapri = new String (""+kp.getPrivate()).getBytes();

   fd_pub.write ( tb_rsapub );
   fd_pri.write ( tb_rsapri );

   fd_pub.close();
   fd_pri.close();
}
```

reccional con trampa propuesto por *Diffie* y *Hellman*, que permite realizar el descifrado al poseedor de la clave secreta (trampa) y de no conocerla, representa la resolución de un problema con un elevado coste computacional. Por ejemplo, los algoritmos *RSA* y de la mochila son de este tipo.

Los criptosistemas de clave pública se caracterizan por utilizar dos claves diferentes

Los criptosistemas de clave pública tienen ciertas ventajas sobre los de clave privada, a destacar por su importancia, el intercambio de claves de sesión, ya que el número de claves que se va a utilizar es más reducido.

Si se tienen *n* usuarios que quieren comunicarse entre sí mediante claves comunes de sesión, con un criptosistema de clave privada habría que disponer de *n.(n-1)/2* claves, mientras que con los sistemas de clave pública únicamente de *2.n*, ya que no es peligroso que el resto de usuarios conozcan la clave pública de los demás, más bien es imprescindible para su comunicación.

El proceso de utilización de los pares de claves se basa en un sencillo proceso. Cada usuario tiene un par de claves E_k, o clave pública y D_k o clave privada o secreta. La clave pública se utilizará para el cifrado, mientras que la privada para el descifrado.

Para poder realizar cualquier tipo de intercambio de información de forma segura, se realizan los siguientes procesos:

- Se cifra un mensaje con la clave pública del destinatario. Por tanto, dando un mensaje *M* y una clave pública E_k, se realiza:

```
C = Ek( M ),
```

siendo *C* el resultado del cifrado.

- El resultado cifrado, *C*, se envía al destinatario, y en estos momentos, únicamente éste puede descifrarlo, ya que mantiene en secreto la clave privada (recuerde privada o trampa).

- Una vez recibido *C* en el destinatario, éste utilizada la clave privada D_k, de tal forma que se obtiene el mensaje *M* original de la siguiente forma:

```
M = Dk (C)
```

Esto se puede resumir en que:

```
M = Dk( Ek( M ) )
```

Lo que implica que un usuario también puede cifrar un mensaje con su clave privada (que normalmente utiliza para el descifrado) y recuperarse el mensaje utilizando la clave pública.

Esta propiedad permite utilizar lo que se denomina firma digital, que garantiza la autenticidad del emisor de un mensaje, como se mostrará más adelante, como si se tratase de su firma manuscrita.

Los criptosistemas de clave pública se caracterizan por utilizar dos claves diferentes

La firma digital es importante ya que es una forma de validar al emisor de un determinado mensaje, recuerde "no repudio", ésta no se puede falsificar y resulta fácil de validar.

```
RSAKey(10001,791a25fcc0541676fd770cab43b473d369606c15ea446e4eef71092aa97944d0bb5
484bbbed218ba04193841c0c3fa34f55e6d23270a91b8dd2ff9c0c152f3cd,pub)

RSA Privada :
RSAKey(6b4b0f7cefa9907dbbf34df62ef621e89ce420df09bfb73b9848bda65ea062a694018c5b4
4b425d9a3ad079fb5a6343064c60868d05739161ba7d93cf27b66d,791a25fcc0541676fd770cab4
3b473d369606c15ea446e4eef71092aa97944d0bb5484bbbed218ba04193841c0c3fa34f55e6d232
70a91b8dd2ff9c0c152f3cd,b82f47bb669d915b552d74c6520d402ca630c45d82c3037d5a191016
375fe233)

Texto en claro :
Prueba - Java. Esto es una prueba de RSAEsto es una prueba de RSA
RSA pub cifrado :
1ffd106d84ad4fb7c893b1b60ee3d93ec5466745e530478861458414 7f2e058f2d16e94fa7a75258
bb0e1f75df4cb3a33991af1159fa2b4f31731100ddf65a3b00

RSA pri descifrado
Prueba - Java. Esto es una prueba de RSAEsto es una prueba de RSA
```

Figura 2.- Ejecución del algoritmo *RSA*.

FUNCIONES UNIDIRECCIONALES CON TRAMPA

Según describen *Diffie* y *Hellman*, las funciones unidireccinales con trampa son aquellas que poseen las siguientes características:

- Son funciones con un dominio definido, de la forma $y=f(x)$, fáciles de calcular para un entero x.

- Existe una función inversa $f^{-1}()$, tal que $x= f^{-1}(y)$.
- De conocer únicamente $y=f(x)$, es imposible computacionalmente encontrar $f^{-1}()$, a menos que se disponga de información adicional.

- Se cumple que $x=f(f^{-1}(x))$, como es de esperar.

Gracias a estas propiedades se disponen de algoritmos como *RSA*, que se basan, por ejemplo, en utilizar como función la resolución de un problema complejo que es el de la factorización de grandes números, como se mostrará a continuación.

Cabe destacar, que a diferencia de las funciones unidireccionales con trampa, las funciones unidireccionales no permiten el cálculo de la función inversa $f^{-1}()$ y tampoco el uso de información adicional para su predeterminación, como ocurre en las primeras.

CIFRADOS EXPONENCIALES

Se basan en el uso de la operación de exponenciación sobre campos finitos como medio para realizar transformaciones criptográficas. El proceso que se realiza al utilizar este tipo de cifrados es el siguiente:

- Sea un mensaje *M* Œ *{0, n-1}* siendo *n* un entero positivo, se tiene entonces que la función de cifrado es:

```
C = Me mod n
```

donde *e* y *n* constituyen los valores de la clave de cifrado.

- La operación de descifrado para conseguir obtener *M* es la siguiente:

```
M = Cd mod n
```

Siendo el par *(d,n)* la clave de descifrado.

A partir de estos conceptos se construyen los distintos algoritmos existentes, como *RSA*, *Pohling-Hellman*, etc., que se muestran a continuación.

ALGORITMO DE POHLING-HELLMAN

Si bien el algoritmo de *Pohling-Hellman* no es un algoritmo de clave privada, porque se utilizan dos claves diferentes para las operaciones cifrado y descifrado, tampoco es estrictamente un algo-ritmo de clave pública, puesto que ambas claves, tanto la del cifrado como la utilizada en el descifrado deben mantenerse en secreto.

La función de cifrado es:

```
C = Me mod n
```

Para el descifrado se realiza lo contrario:

```
M = Cd mod n
```

Donde *n* es un número primo muy grande (observe que no se indica que es construido como el resultado de otros dos números primos). Se cumple a su vez que *e.d = 1 mod n.*

Toda la aritmética se hace en un *campo de Galois CG(n)* y como *n* es primo, el *indicador de Euler* que se tiene es *F(n)= n-1*. Si alguien conociera *e* y *n*, tendría que calcular *d*, pero sin el conocimiento de *e o d,* debe resolver el siguiente problema:



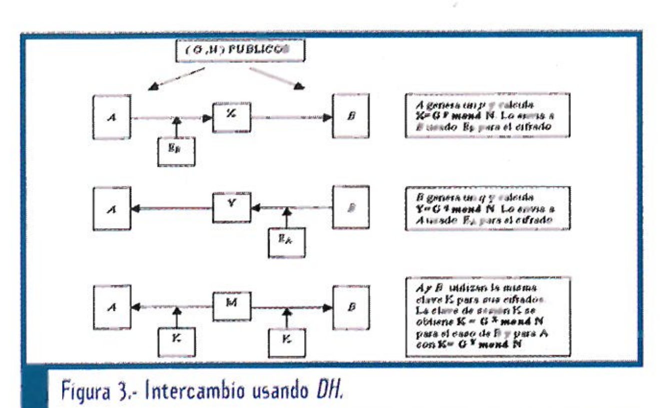


Figura 3.- Intercambio usando *DH*.

e = logMC mod n,

que actualmente es bastante complicado.

Este tipo de algoritmo corresponde a un criptosistema convencional, ya que las claves *e* y *d* deben permanecer en secreto y por tanto no es de clave pública, aunque por su funcionamiento siempre se muestra como uno perteneciente a ese tipo de algoritmos.

Veamos un sencillo ejemplo de funcionamiento:

- Sea $n = 11$, primo, por tanto $F(11) = 10$, y las claves $d = 7$ y por tanto e = inverso (7,10)= 3.
- Si $M = 5$, se cifra $C = 5^3 \, mod \, 11 = 4$.
- Se descifra, $M = 4^7 mod \, 11 = 5$.

Como puede comprobar, todo el proceso es bastante repetitivo, aunque con un coste computacional de cálculo algo elevado.

ALGORITMO RSA

El algoritmo *RSA* de *Rivert, Shamir y Adleman* está basado en el problema de la factorización de grandes números. A su vez, este problema se relaciona con el de determinar si un número con un tamaño elevado es primo (por ejemplo 200 dígitos), y por tanto se complementan.

Para generar dos claves, se eligen dos números aleatorios muy grandes y primos que se denominan *p* y *q*. Para una mayor seguridad, se determinan *p* y *q* con la misma longitud. Se calcula el siguiente producto:

n = p.q

A continuación, se determinan la clave pública *e* mediante un número aleatorio, tal que *e* y *(p-1)(q-1)* son primos entre sí. Recordar que *p* y *q* deben mantenerse en secreto, y aunque no se utilizarán posteriormente, es importante no darlos a conocer.

Para obtener la clave privada *d* se aplica el algoritmo de *Euclides*, por lo que se tiene que:

LISTADO 2. Utilización claves RSA.

```
public void usoRSAKeys() throws Exception
{
  KeyPair kp;
  CipherKey dk = null ;
  String cs_key = null, ds_key = null ;

 DataInputStream fd_pub = new DataInputStream
     ( new FileInputStream ("rsapub"));
 DataInputStream fd_pri = new DataInputStream
     ( new FileInputStream ("rsapri"));

 cs_key = fd_pub.readLine();
 ds_key = fd_pri.readLine();

 CipherKey c_key =
     (CipherKey) Crypto.fromString ( cs_key );
 CipherKey d_key =
     (CipherKey) Crypto.fromString ( ds_key );

 fd_pub.close();
 fd_pri.close();

 String mensaje = "esto es una prueba";
 StringBuffer sb = new StringBuffer("");

 for ( int cont_car = 0 ;
      cont_car < (64 - mensaje.length()) ; cont_car++ )
         {
             sb.append(" ");
         }

mensaje += sb.toString();

System.out.println ( mensaje.length());
byte [] claro =  mensaje.getBytes();
byte [] cifrado = new byte [ mensaje.length() ] ;

c_key.encrypt ( claro, 0, cifrado, 0 );
System.out.println ( "RSA pub cifrado \n\n\n" +
Crypto.hexString( cifrado));

d_key.decrypt ( cifrado, 0, claro, 0 );
System.out.println ( "RSA pri descifrado \n\n\n" +
new String(claro) );
}
```

```
d = e-1 mod ((p-1)(q-1))
```

Obsérvese que *d* y *(p-1)(q-1)* deben ser primos entre sí.

A partir de este momento ya se dispone de las claves pública y privada respectivamente. La pública esta compuesta por el par *(e,n)* y la privada por *(d,n)*.

Mediante *e* y *d* se pueden realizar las operaciones de cifrado y descifrado que cumplen las propiedades de funciones unidireccionales con trampa.

Para cifrar un mensaje *M*, éste se divide en bloques de tamaño menor que *n*. Esto significa, que si *p* y *q* son dos números de 100 dígitos, *n* tendrá 200 dígitos y por tanto, cada bloque a cifrar debería tener a lo sumo 200 dígitos. Como siempre, si se utilizan bloques de tamaño fijo siempre se puede realizar un relleno con ceros a la izquierda.

Veamos un ejemplo numérico de funcionamiento del algoritmo y continuación se mostrará como utilizar la librería *Cryptonite* para realizar cifrados mediante *RSA*.

Tomemos por ejemplo *p=47* y *q=71*, entonces se tiene que:

```
n = p.q = 47.71=3337
```

A — E_A — B; E_B
A envía su clave pública (E_A) usando la clave pública de *B* (E_B) para el cifrado

A — K_S — B; E_A
B genera una clave de sesión y la envía a A usando la clave pública E_A

A — K_S; D_A
A utiliza su clave privada para obtener así la clave de sesión K_s.

Figura 4.- Intercambio de claves alternativo.

Debemos escoger como clave pública *e* un número sin factores en común con el siguiente:

```
(p-1) (q-1) = 46 . 70 = 3220
```

por ejemplo se puede determinar *e* con el valor **79**.

Para calcular la clave privada, calculamos el inverso módulo *(p-1)(q-1)*, luego:

```
d = 79-1 mod 3220 = 1019
```

que puede ser calculado fácilmente utilizando el algoritmo de *Euclides*.

Una vez realizado este proceso, se dispone de la clave pública, *e*, cuyo valor es **79** y de la privada *d*, cuyo valor es **1019**. A su vez *n* debe ser público y vale **3337**.

Para cifrar el siguiente mensaje: **688232687966668003**, se divide éste en bloques de tres dígitos, por lo que se obtienen seis partes para cifrar. Si el primer bloque *m1* tiene el valor **688**, el cifrado sería:

```
C1 = 68879 mod 3337 = 1570
```

Realizando la misma operación con los siguiente bloques se obtiene que el mensaje cifrado corresponde a:

```
C = 1570 2756 2091
2276 2423 158
```

El proceso de descifrar el mensaje se haría de la misma manera, pero utilizando esta vez la clave privada, con lo que se tendría:

```
d1=15701019 mod
3337=688 = m1
```

Repitiendo este proceso con el resto de los bloques se compondría el mensaje inicial.

Una vez que los conceptos teóricos se han mostrado, será mejor pasar a la sencilla parte práctica, ya que como la implementación de todo ese proceso viene dada por la librería que se está utilizando, de tal forma que sólo debemos tener claro el proceso de utilización.

En el siguiente ejemplo, Listado 1, vamos a crear las claves pública y privada, que almacenaremos en dos archivos diferentes, para luego cifrar una cadena de texto con la clave pública almacenada en el fichero correspondiente, y una vez realizado el cifrado, completar el descifrado utilizando la clave privada que se ha almacenado.

Actualmente RSA se ha mostrado con el mejor algoritmo de clave pública

La primera parte que tenemos que destacar se refiere al uso de una referencia del tipo *KeyPair kp*, mediante la cual podemos obtener la clave pública y privada que generemos.

Para generar un par de claves *RSA* de un determinado tamaño, únicamente se debe realizar una llamada a la función *RSAKey.createKeys()*, pasándole como parámetro el tamaño de la clave, en este caso, 512 *bits*. El conjunto generado se asigna a *kp*, por lo que podemos obtener así las claves privada y pública por separado. Para ello se invocan a los métodos *getPublic()* y *getPrivate()* con *kp*, obteniéndose el *array* de *bytes* que corresponda.

Para almacenar las claves únicamente debemos disponer de dos objetos *FileOutputStream* diferentes, y mediante la llamada al método *write()*, pasándole como paráme-



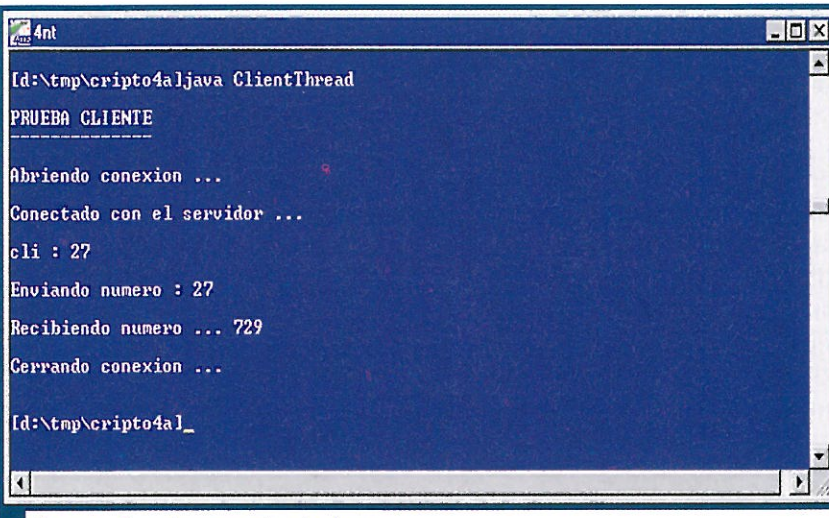


Figura 5.- Ejemplo intercambio de cliente.

tro el *array* de *bytes*, escribirlos en distintos ficheros, por lo que ya disponemos de la clave pública y privada para utilizarla a nuestra conveniencia.

En el Listado 2 se muestra cómo utilizar los ficheros con las claves generadas. El proceso es bastante simple. Mediante la creación de los correspondientes *CipherKey* para cada clave, se les asigna contenido mediante la ejecución de la llamada *Crypto.fromString()*. En ese momento ya se dispone de las claves pública y privada lista para utilizarse. A continuación se genera un *buffer* con los datos a cifrar de un tamaño determinado. El tamaño se puede determinar mediante la función *cipherBlockSize()*.

Una vez que se dispone del mismo, se realizan llamadas a la función de cifrado mediante la clave pública, *c_key* y la función, *c_key.encrypt()*.

Para realizar el descifrado se utiliza la clave privada, *d_key* y la invocación a *d_key.decrypt()*. Como se puede apreciar en el listado, el manejo de las claves y su gestión es muy sencillo y fácil.

En la Figura 2 se puede apreciar la ejecución del Listado 1 y 2 sobre un texto en claro aleatorio. Modifique los fuentes para adaptarlos a sus necesidades.

DIFFIE-HELLMAN

El algoritmo de *Diffie-Hellman* fue el primer algoritmo de clave pública desarrollado. Su seguridad se basa en el cálculo de logaritmos en campos finitos, en contra de los cifrados exponenciales.

La propiedad interesante de este algoritmo es que puede ser utilizado para la distribución de claves, esto es, determinar una clave de sesión, pero no para el cifrado y descifrado de mensajes.

Su utilización es relativamente sencilla. Sean A y B dos personas que quieren ponerse en contacto para utilizar una clave única común, es decir, ambos usarán un criptosistema de clave privada. Para ello, A y B generan dos números enteros suficientemente grandes, n y g tal que g es menor que n pero más grande que 1. Estos números no tienen por qué ser secretos, sino que pueden acordarse en un canal inseguro. Entonces se siguen los siguientes pasos:

- A elige un número entero x aleatoriamente y calcula $X = g^x \mod n$.
- B elige otro número aleatorio y calcula: $Y = g^y \mod n$.
- A envía X a B y B envía a A el resultado Y. Obsérvese que A mantiene en secreto x y B a su vez y.
- A calcula $k = Y^x \mod n$.
- B calcula $k' = X^y \mod n$.

Por tanto, k y k' son iguales a $g^{xy} \mod n$. La elección de unos g y n adecuados puede determinan la seguridad del sistema.

En la Figura 3 puede apreciarse el proceso de intercambio de claves de forma gráfica, que puede aclarar todo el proceso.

Seguramente el lector habrá notado que existe una alternativa más sencilla para el intercambio de claves de sesión que *Diffie-Hellman*, y es utilizar la misma semántica de los criptosistemas de clave pública. El proceso sería el siguiente:

- Un sujeto A envía a otro B su clave pública. Si es dentro de un canal seguro, mejor.
- El sujeto B genera una clave de sesión aleatoria k, que cifra usando la clave pública de A que acaba de recibir.
- A descifra el mensaje usando para ello su clave privada, por lo que A y B ya conocen la misma clave de sesión.

En la Figura 4 se puede observar todo este proceso gráficamente.

Existe una gran variedad de algoritmos para el intercambio de claves, pero observando el anterior, uno se percata de lo sencillo que puede llegar a ser el proceso.

Gracias a la seguridad que ofrece RSA, éste suele ser el preferido para realizar aplicaciones importantes

Para comprobar el uso del intercambio de claves mediante *Diffie-Hellman* se va a mostrar un ejemplo práctico en *Java* mediante el cual un programa cliente y otro servidor se intercambian un par de números. Pruebe a modificar el código para utilizar cadenas de caracteres.

En el Listado 3, incluido en el *CD-ROM*, se puede observar la parte cliente. La primera parte interesante es cómo se determina el *stream* de datos, de forma que su generación involucre todo el desarrollo del algoritmo de *Deffie-Hellman* y el intercambio de una clave para utilizar *TripleDES* de 512 *bits* (desarrollado en el segundo artículo), mediante el modo *OFB* (abordado en el tercer artículo).

Todo esto se realiza mediante una única sentencia en el procedimiento *makeCS()* que devuelve un objeto tipo *CipherStreamClient* a través del cual serán manipulados los datos.

Como la ejecución de estos procedimientos está preparada para poder desarrollarse en un *thread*, el método *run()* contiene la conexión mediante *Socket()*. A continuación se crean los flujos de datos, siendo el más importante el que se realiza mediante *CipherStreamClient,* ya que a partir de éste el resto de flujos del *socket*, tanto el de entrada como el de salida, funcionarán bajo su control, esto es, cifrado mediante *TripleDes* y con la clave de sesión determinada mediante el protocolo *Deffie-Hellman.*

Una vez preparado todo esto, enviamos un número al servidor y esperamos a recibir otro, que es visualizado. Así de sencillo. En la imagen 5 se muestra un ejemplo de ejecución de la parte cliente.

En el Listado 4, también incluido en el *CD-ROM*, se muestra la parte de servidor. Es muy simple, lo que ayuda a comprender mejor todo este proceso. El proceso es básicamente el mismo que ocurre en el lado cliente, como es de esperar, sólo que la semántica para utilizarlo es diferente.

En el método *makeCS* se recibe un *socket* y se devuelve un objeto tipo *CipherStreamServer* que determina el tratamiento de los datos que se gestionen a través del *socket*. Al igual que en la parte cliente, determinamos el uso de *TripleDES* y modo *OFB*.

```
4NT Prompt
[d:\tmp\cripto4a]java ServerThread
PRUEBA SERVIDOR
---------------
Esperando conexion ...
Conexion realizada ...
Esperando numero  ...
Se ha recibido : 27
Enviando de vuelta ... 729
Finalizada la conexion

[d:\tmp\cripto4a]
[d:\tmp\cripto4a]
```

Figura 6.- Ejemplo intercambio servidor.

En el método *run()* se realiza el mismo proceso que en el cliente, y éste podría ser adaptado para atender múltiples peticiones fácilmente.

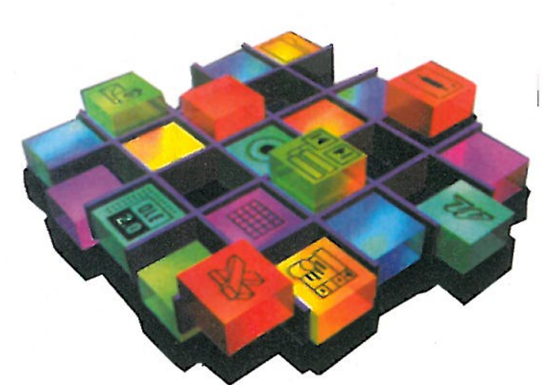
La diferencia más importante es el uso de *CipherStreamServer* para realizar el proceso en el lado servidor. El resto sigue el mismo proceso, se recibe un entero, se eleva al cuadrado y se envía al cliente. Práctico, sencillo y muy fácil de programar, tal y como se puede apreciar en el ejemplo de la Figura 6.

Como se puede apreciar, el uso del protocolo/algoritmo de *Deffie-Hellman* para el intercambio de claves de sesión es sencillo y práctico, pero en determinadas circunstancias es necesario utilizar otros protocolos que ofrezcan una solución a determinadas cuestiones, como por ejemplo, ¿quién me dice a mí que el servidor al que estoy conectado es válido y no es uno ilegal puesto para capturar datos?. Piense sobre ello un momento y determine qué cambios se podrían realizar para remediarlo.

En el siguiente artículo se presentará *ElGamal*, un algoritmo asimétrico ampliamente reconocido y utilizado, con lo que se dará por finalizada la descripción de este tipo de algoritmos. Se mostrará además la firma digital y mediante un sencillo ejemplo el lector podrá comprobar su importancia práctica.

CONCLUSIONES

Se han presentado y descrito el conjunto de algoritmos de clave pública más renombrados y con mayor número de implementaciones desarrolladas en *Java*. Ya que éstos están sobradamente probados, no parece muy probable que existan puertas traseras o similares. Mediante los ejemplos desarrollados, el lector puede aprender a utilizar estos algoritmos en sus aplicaciones de forma rápida, eficaz y sencilla.



Suite para Internet (V): La transmisión de ficheros

Jordi Agost Moré.
Profesor de Programación/Multimedia de la Universidad de Lleida.

¿Cómo podemos recibir ficheros de un sitio *Web* concreto? ¿Cómo podemos hacer que nuestra aplicación envíe ficheros de control o sobre su estado a un servidor de *Internet*? ¿Cuáles son los pasos que debemos seguir? ¿Cómo lo podemos realizar con el menor número de controles posible? A éstas y otras preguntas intentaremos responder a través de este artículo.

Una vez abordadas las aplicaciones más importantes de *Internet* vamos a ver una utilidad que aunque sea menos popular que otras como la *Web* o el correo electrónico, no es menos útil. Nos referimos al envío y transmisión de ficheros a través de la red.

EL CONTROL INTERNET TRANSFER

Para ello vamos a utilizar en primer lugar el control *Internet Transfer* de *Microsoft*.

Dicho control ofrece una implementación de dos de los protocolos que hoy en día más se utilizan en *Internet*. Estamos hablando del protocolo de transferencia de hipertexto, conocido comúnmente por sus siglas *HTTP (Hypertext Transfer Protocol)* y el protocolo de tranferencia de archivos, también conocido por sus siglas *FTP (File Transfer Protocol).*

El primer protocolo, *HTTP*, lo utilizaremos para conectarnos con los servidores *Web* y de esta forma obtener páginas *HTML*. Podríamos decir que dicho protocolo lo utilizamos para pedir la información al servidor.

Con el segundo protocolo podremos iniciar sesiones en servidores *FTP* para, de este modo descargar y cargar los archivos que creamos conveniente.

OTRAS APLICACIONES POSIBLES CON INTERNET TRANSFER

Vamos a desarrollar en primer lugar un explorador *FTP*, pero debemos tener en cuenta que ade-

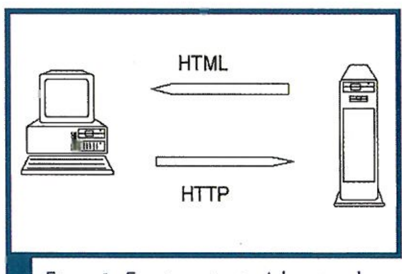


Figura 1.- Funcionamiento del protocolo *HTTP* utilizado para hacer peticiones a un servidor *Web*.



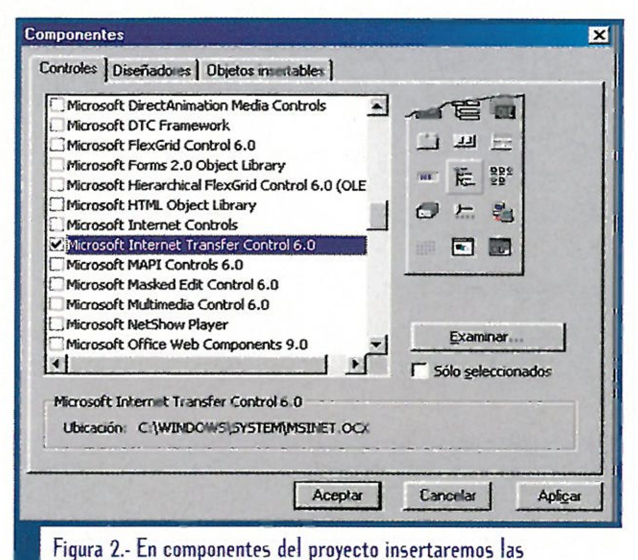


Figura 2.- En componentes del proyecto insertaremos las referencias para activar el control.

más de la aplicación mencionada también podemos utilizar el control *Internet Transfer* para otras tareas, como por ejemplo, un analizador de páginas *Web*, llamado comúnmente "araña" o *spider*, cuyo fin es explorar un sitio concreto de la *World Wide Web* y coger tan sólo la información que deseemos (por ejemplo los gráficos de ese sitio *Web*). Podríamos también realizar una aplicación que transfiriese o enviase archivos a un determinado sitio *FTP*.

ESQUEMA BÁSICO DEL FUNCIONAMIENTO DEL CONTROL INTERNET TRANSFER

El funcionamiento del control *Internet Transfer* al poseer la implementación de dos protocolos distintos, es muy diversa y dependerá en gran medida del protocolo que elijamos.

Aún así hay una serie de pasos que deberemos seguir independientemente del protocolo de comunicación que utilicemos, dichos pasos son los siguientes:

1. Establecer la propiedad *AccessType* a un servidor *proxy* correcto.

2. Invocar el método *OpenURL* con una dirección *URL* válida.

3. Invocar el método *execute* con una dirección *URL* válida y con un comando adecuado para el protocolo que estemos utilizando.

4. Utilizar el método *Get Chunk* para recuperar datos del *buffer*.

Veamos ahora un poco más detenidamente cada uno de estos pasos necesarios para el funcionamiento.

ESTABLECIENDO LA PROPIEDAD ACCESSTYPE

El primer paso para establecer una comunicación con *Internet* será conocer la forma en que el equipo que desea establecer la conexión se encuentra conectado a *Internet*. Si dicho equipo se encuentra conectado a una *intranet* lo más seguro es que su conexión se realice a través de un servidor *proxy*.

Para la comunicación necesitaremos saber si nuestra conexión a Internet se realiza a través de un servidor proxy

Un servidor *proxy* es un equipo que actúa como intermediario entre nuestro equipo e *Internet*. Su misión básica es la de un servidor de seguridad ya que se encarga por una parte de descartar las peticiones de usuarios finales no válidos y por otra parte se encarga de descartar las peticiones externas para proteger la *intranet* de acciones hostiles.

CÓMO CONOCER LA CONFIGURACIÓN PROXY

Para conocer la configuración *proxy* de nuestro equipo deberemos seguir una serie de sencillos pasos:

1. Debemos hacer un *clic* en **Inicio->Configuración->Panel de control.**

2. Hacemos un *doble clic* en el icono de *Internet*.

3. En las propiedades de *Internet* elegimos el botón de conexión.

4. Miramos si está activada la casilla de verificación **Conectar a través de un servidor proxy**.

5. En caso afirmativo miramos la opción **Configuración.**

Utilizaremos el control Internet Transfer para implementar los protocolos FTP y HTTP

Si queremos que nuestro programa acceda a *Internet* utilizando un servidor *proxy* diferente del que está predeterminado entonces lo que haremos será establecer las propiedades *Proxy* y *AccessType*:

```
Inetctr.Proxy = Nuevo_Nombre
Inetctr.AccessType = icNamedProxy
```

Donde **Nuevo_Nombre** es una cadena que identifica el nombre del *proxy* y *AccessType* y que tiene uno de los valores que se observan en la Tabla 1.

INVOCANDO EL MÉTODO OPENURL

Una vez que hemos especificado la propiedad *AccessType*, lo siguiente es utilizar el método *OpenURL*, por supuesto con una dirección *URL* correcta. El resultado

TABLA 1. Estructura TNotifIcon.

Constante	Valor	Descripción
IcUseDefault	0	Se utilizan las opciones predeterminadas del registro para el acceso a Internet.
IcDirect	1	El control tiene acceso directo a Internet.
IcNamedProxy	2	Le indicamos al control que use el proxy indicado en la propiedad Proxy.

dependerá de la dirección *URL* de destino. Si dicha dirección contiene una página *Web*, ésta será devuelta, pero si contiene un archivo de texto se obtendría el archivo real. Así si por ejemplo asignamos a una caja de texto la siguiente sentencia:

```
Inetctr.OpenURL(ftp://ftp.micro-
    soft.com/disclaimer.txt)
```

Obtendremos el contenido del archivo indicado en la caja de texto. Pero la conexión que se habrá realizado es síncrona, lo que significa que la operación de transferencia se produce antes de que se ejecute cualquier otro procedimiento y la transferencia de datos ha de terminar obligatoriamente antes de ejecutar algún otro tipo de código.

EL MÉTODO EXECUTE

Para solucionar este inconveniente deberemos utilizar el método *Execute* que nos permitirá ejecutar otro código mientras los datos se van recibiendo en un segundo plano.

A través del método Execute lograremos una comunicación asíncrona

En este caso debemos tener en cuenta que en este proceso los datos no son redireccionados directamente como antes, sino que se sitúan en un *buffer* intermedio. Después veremos cómo recuperar dichos datos a través del método *GetChunk*.

En la Tabla 2 podemos observar los diferentes comandos de *FTP* que podemos ejecutar (mediante el método *Execute*) a través del control *Internet Transfer*.



Figura 3.- Configuración del sistema *Windows* de un servidor *proxy*.

UN CLIENTE FTP

Vamos a ver ahora un ejemplo más concreto de la utilización del control *Internet Transfer*. En dicho ejemplo desarrollaremos un programa que será capaz de recibir ficheros desde un determinado servidor *FTP* al que accederemos a través de un nombre de usuario y una contraseña.

LA CONEXIÓN AL SERVIDOR FTP

La función siguiente es la que se ejecutará cuando pulsemos el botón *Conectar*:

```
Private Sub cmdConectar_Click()

Dim varTmp As Variant
    Itr1.URL =
    txtDireccionFTP.Text
    Itr1.UserName =
    txtUsuario.Text
    Itr1.Password =
    txtContraseña.Text
    Itr1.Execute , "DIR"
    varTmp = Itr1.GetChunk(1024)
    subShowFiles (varTmp)

End Sub
```

En ella, lo primero que hacemos es asignar a la propiedad *URL* el valor de la caja de texto que indica la dirección *FTP* donde nos vamos a conectar. Con el mismo procedimiento asignamos el nombre de usuario y la contraseña. A continua-



TABLA 2. Comandos FTP admitidos por el control Internet Transfer.

Operación	**Descripción**
CD archivo1	Cambiar directorio. Cambia al directorio especificado en el archivo1.
CDUP	Cambiar al directorio superior. Equivale a "CD.."
CLOSE	Cierra la conexión FTP actual.
DELETE archivo1	Elimina el archivo especificado en archivo1.
DIR archivo1	Directorio. Busca en el directorio especificado en archivo1. Se admiten comodines, pero el host remoto determina la sintaxis. Si no especifica archivo1, obtendrá una lista completa del directorio de trabajo actual.
	Puede usar el método GetChunk para obtener los datos del directorio.
GET archivo1 archivo2	Recupera el archivo remoto especificado en archivo1 y crea el nuevo archivo local especificado en archivo2.
LS archivo1	Lista. Busca en el directorio especificado en archivo1. Se admiten comodines, pero el host remoto determina la sintaxis. Puede usar el método GetChunk para obtener los datos de los archivos del directorio.
MKDIR archivo1	Crear directorio. Crea el directorio especificado en archivo1. El éxito de la operación depende de los privilegios del usuario en el host remoto.
PUT archivo1 archivo2	Copia el archivo local especificado en archivo1 en el archivo del host remoto especificado en archivo2.
PWD	Mostrar directorio de trabajo. Devuelve el nombre del directorio actual. Puede usar el método GetChunk para obtener los datos.
QUIT	Termina la sesión del usuario actual.
RECV archivo1 archivo2	Recupera el archivo remoto especificado en archivo1 y crea un nuevo archivo local especificado en archivo2. Equivale a GET.
RENAME archivo1 archivo2	Cambia el nombre del archivo indicado en archivo1 por el nombre especificado en archivo2. El éxito de la operación depende de los privilegios del usuario en el host remoto.
RMDIR archivo1	Eliminar directorio. Elimina el directorio remoto especificado en archivo1. El éxito de la operación depende de los privilegios del usuario en el host remoto.
SEND archivo1 archivo2	Copia el archivo local especificado en archivo1 en el archivo del host remoto especificado en archivo2. Equivale a PUT.
SIZE archivo1	Devuelve el tamaño del directorio especificado en archivo1.

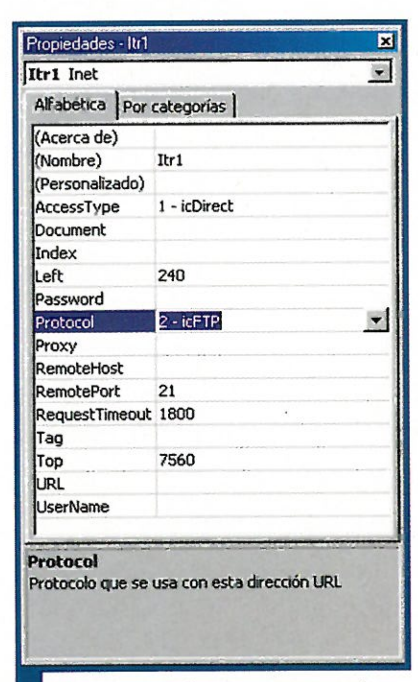


Figura 4.- Pantalla de propiedades del control *Internet Transfer*, donde observamos la elección del protocolo.

ción ejecutamos el comando *FTP* **"DIR"**, que buscará los archivos en el directorio de trabajo actual y seguidemante utilizaremos el método *GetChunk* para obtener los diferentes datos.

RECUPERANDO ARCHIVOS

En la rutina *MostrarFicheros* lo que hacemos es recuperar los ficheros del directorio actual. En primer lugar declaramos dos variables, la primera llamada **strArray** que se utiliza para almacenar los diferentes ficheros, y posteriormente una variable que nos servirá como contador. Seguidamente llenaremos el *array* de nombres de ficheros con la ayuda de la función *Split:*

```
strArray = Split(CStr(var),
 Chr(13) & Chr(10))
```

Dicha función cogerá la cadena pasada como argumento y la dividirá en diferentes elementos del *array* separándolos por los caracteres de retorno de carro (carácter **13** más carácter **10** o en código: Chr(13) & Chr(10)). Añadimos el símbolo para ir al directorio superior (para sistemas que no son *UNIX*) y luego con un sencillo bucle vamos a llenar la lista o *Listbox*. Veamos ahora toda la subrutina al completo:

```
Public Sub MostrarFicheros(var As
    Variant)
  Dim strArray() As String
  Dim intTmp As Integer
   lstFicheros.Clear

   strArray = Split(CStr(var),
     Chr(13) & Chr(10))
   lstFicheros.AddItem ("../")
   For intTmp = 0 To
     UBound(strArray)
       lstFicheros.AddItem
     (strArray(intTmp))
   Next
End Sub
```

Vamos ahora a examinar la rutina que constituye el corazón de todo el sistema y que es la que se ejecutará cuando hagamos un *doble clic* sobre la lista de ficheros:

```
Private Sub
    lstFicheros_DblClick()

On Error GoTo etiquetaerror

Dim strFile As String
Dim booTmp As Boolean

 If (Left(lstFicheros.Text, 2) =
     "./" Or
      Left(lstFicheros.Text, 3) =
     "../" Or
      Right(lstFicheros.Text, 1) =
     "/") Then
       Itr1.Execute , "CD " &
     lstFicheros.Text
       Itr1.Execute , "DIR"
       MostrarFicheros
     (.GetChunk(1024))
 Else
   booTmp = True
   strFile = NombreAEnviar
   Itr1.Execute , "SIZE " &
```

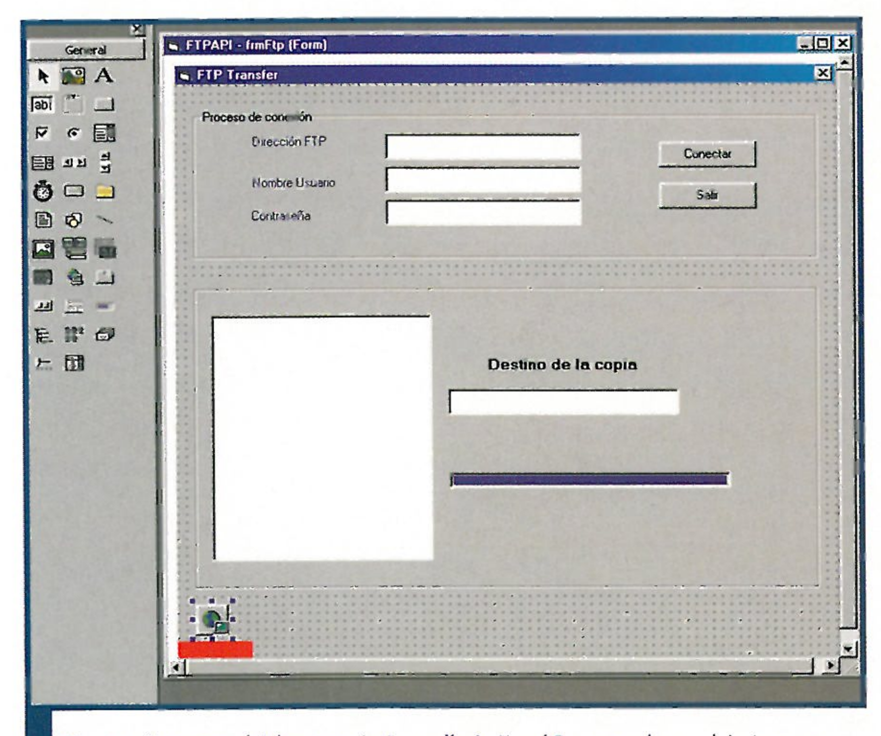


Figura 5.- Vista parcial del entorno de desarrollo de *Visual Basic* con el control de *Internet Transfer* visible en tiempo de diseño.



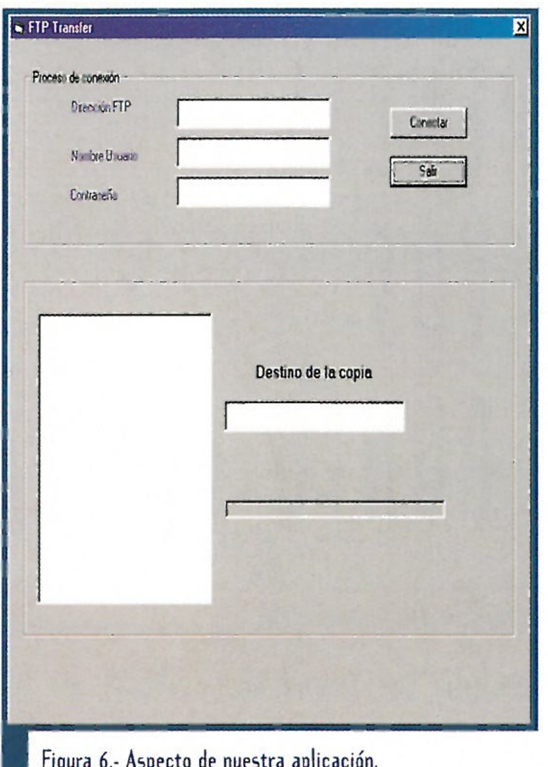


Figura 6.- Aspecto de nuestra aplicación.

```
    lstFicheros.Text
   prbl.Max = .GetChunk(1024)
   Itrl.Execute , "GET " &
    lstFicheros.Text & " " &
    strFile
   Itrl.Execute , "PWD"
   MsgBox "Final de la transfe-
    rencia de archivos"
   prbl.Value = 0
   booTmp = False
 End If

etiquetaerror:
    If Err.Number = 35764 Then
        DoEvents
        If booTmp And Not
    (Dir(strFile) = "") Then
            prbl.Value =
    FileLen(strFile)
        Resume
    End If
End Sub
```

En ella vemos que lo primero que hacemos es fijar un tratamiento de errores que, como veremos posteriormente, resultará de gran utilidad. Después miramos si la cadena o miembro de la lista *lstFicheros* sobre la que hemos hecho el *doble clic* es un directorio. Lo sabremos porque en el caso de que lo sea poseerá los caracteres "./", "../" o bien "/". Entonces utilizando el método *Execute*, enviaremos a través del control el comando *CD* para cambiar de directorio y el comando *DIR* para obtener los nombre de los ficheros del directorio de trabajo actual:

```
   Itrl.Execute , "CD "
& lstFicheros.Text
   Itrl.Execute , "DIR"
```

A continuación a través del método *Get Chunk* recuperaremos los archivos:

```
MostrarFicheros
  (.GetChunk(1024))
```

El siguiente paso que se debe seguir (cuyo punto de entrada es la sentencia *Else*) corresponde al caso en que hagamos hecho *doble clic* sobre un fichero y tengamos la necesidad de recuperarlo. Mediante la orden *SIZE* del protocolo obtendremos su tamaño:

```
Itrl.Execute , "SIZE " &
  lstFicheros.Text
```

Dicho tamaño lo recuperaremos, como siempre, con el método *GetChunk* y servirá para establecer el límite máximo de la barra de progreso que mostraremos:

```
Itrl.Execute , "DIR"
MostrarFicheros
  (.GetChunk(1024))
```

A continuación obtendremos los ficheros con la orden *GET*, y seguidamente mediante la orde *PWD* provocaremos un error que podamos interceptar. Este error producido por el hecho de que aún no podremos ejecutar el siguiente comando hasta que no acabe la orden *GET* nos servirá para poder actualizar la barra de progreso que nos indica la evolución de nuestra descarga de ficheros.

```
Itrl.Execute , "GET " &
 lstFicheros.Text & " " &
 strFile
Itrl.Execute , "PWD"
```

En la ejecución de este tratamiento de errores comprobaremos que el error sea el número **35764** que significa que aún se está ejecutando el último comando. Entonces, y suponiendo que hayamos bajado algún archivo, miraremos su longitud y actualizaremos la barra de progreso a su valor correspondiente.

```
If Err.Number = 35764 Then
    DoEvents
    If booTmp And Not
(Dir(strFile) = "") Then
prbl.Value = FileLen(strFile)
    Resume
End If
```

Y como punto final debemos comentar la función *NombreAEnviar* que determinará el nombre de salida del fichero. Esta función es necesaria porque dentro del lenguaje o protocolo *FTP* pueden existir diversas opciones de sintaxis.

CONCLUSIÓN

Hasta aquí hemos visto cómo realizar un sencillo cliente de *FTP*. El proceso de construcción de dicha aplicación constituye la base de muchas aplicaciones existentes hoy en día y es utilizada normalmente para la transferencia de ficheros.

TRANSFERENCIA DE ARCHIVOS PUNTO A PUNTO (y VI)

Enrique de la Lastra. *Desarrollador Independiente.*

La última fase de la implementación del protocolo de nivel de enlace es la definición de los procesos que hay que realizar en función del estado en que se encuentre el transmisor o el receptor.

INTRODUCCIÓN

En este último terminaremos el componente *Delphi* que implementa el protocolo de nivel 2 que hemos diseñado, y que se apoya en otro componente *Delphi* que implementaba a su vez las funciones básicas del nivel 1 *OSI* (*Open Systems Interconnection,* Interconexión de Sistemas Abiertos).

En el número anterior realizamos el desarrollo del código (tipos, variables y funciones) necesario para delimitar las tramas recibidas y obtener del flujo de *bytes* recibidos, todos los campos de que se compone la trama. Esos campos los almacenamos en una estructura de datos (de tipo **TTrama**, definido por nosotros) para poder realizar después su tratamiento.

En el código implementado quedaron pendientes de desarrollar, detalles aparte, los procedimientos de tratamiento y de envío de los distintos tipos de tramas. Es decir, por un lado, queda por programar el código del procedimiento **ActuarSobreTrama**, en el que, en función de la trama recibida, se decidirá cuál es la reacción que debe llevar a cabo el componente. Por otro lado, debemos desarrollar los procedimientos de envío de las tramas de información y las tramas *ACK* y *NACK* (*Acknowledge* y *Negative Acknowledge*).

Debemos tener en cuenta que el componente es único y por tanto no hay un desarrollo separado de un componente de transmisión y otro de recepción, sino que todo el comportamiento está integrado en un único código. De esta forma, debemos saber siempre en qué estado se encuentra la transmisión, así como saber en cada momento si somos el transmisor o el receptor.

DEFINICIÓN DE LOS ESTADOS

A través de la variable **estadoTransmision** sabemos que, en un momento dado, estamos esperando una trama (**etEsperaDatos**) y por tanto somos el receptor, o estamos esperando un *ACK* (**etEsperaACK**), por lo que nos estamos comportando como el transmisor.

En un principio, los dos extremos de la transmisión están esperando una trama, ya que potencialmente se van a comportar siempre

como receptores. En el momento en que uno de los dos extremos decida transmitir una trama con información, pasará a esperar su correspondiente *ACK* y por tanto pasará al estado **etEsperaACK.**

Si los dos extremos transmiten una trama de información al mismo tiempo, los dos pasarán al estado **etEsperaACK.** Por tanto, si estando en el estado **etEsperaACK** recibimos una trama de información, deducimos que nos hemos decidido a transmitir en el mismo momento que el otro extremo, por lo que, volvemos a pasar al estado **etEsperaDatos.** En este caso, esperamos un tiempo aleatorio y volvemos a transmitir la trama.

Como el tiempo de espera para retransmitir la trama es aleatorio en ambos lados de la comunicación, es fácil que uno comience a transmitir antes que el otro, consiguiendo de esta forma convertirse en el transmisor y consiguiendo, al mismo tiempo, convertir al otro extremo en receptor.

Siempre que enviamos una trama de información, y por tanto nos convertimos en transmisor, arrancamos un *timeout*, para asegurarnos que, en caso de pérdida de la trama de información o de la trama de *ACK*, no nos quedamos indefinidamente esperando el *ACK*. En este caso, después de enviar la trama de información, pasamos al estado **etEsperaACK** y cada vez que arranquemos el *timeout*, seguiremos en ese mismo estado reenviando la última trama enviada.

Si la línea está rota o el receptor está apagado (o fuera de cobertura), seguiríamos reenviando sin parar la misma trama cada vez que venciese el *timeout*. Para evitar que esto ocurra, fijamos una número límite de reintentos, de forma que vencido ese límite, deducimos que la línea está rota.

El límite de reintentos lo fijamos como siempre mediante una constante, para modificarlo fácilmente en el futuro si nos interesa:

```
const
  NUM_MAXIMO_REPETICIONES = 10;
```

Por último debemos contemplar lo que ocurre cuando estando en el estado **etEsperaACK** recibimos una trama. Si la trama recibida es un *ACK*, pasamos de nuevo al estado **etEsperaDatos** y *reseteamos* el *timeout*. Si, por el contrario, la trama recibida es un *NACK*, reenviamos la última trama de información que habíamos enviado y volvemos a arrancar el *timeout*.

Un timeout variable soluciona el caso en que ambos extremos comienzan a transmitir al tiempo

En caso de que esperando una trama de información (**etEsperaDatos**) llegue una trama *ACK* o *NACK*, la única posibilidad es que se haya producido un error, por lo que mediante un evento que luego veremos, informaremos de ello.

Todo el código de actuación necesario para el tratamiento de las tramas, se realiza en el procedimiento **ActuarSobreTrama** que dejamos indicado el mes pasado (dentro del procedimiento **ProcesarTrama**, que realizaba la segmentación de las tramas y la recuperación de los campos de las mismas).

En concreto, encontrándonos en el estado **etEsperaDatos**, y según lo visto, necesitaríamos el siguiente código:

```
case estadoTx of
 etEsperaDatos:
  if ComprobarCRC (---) then
    begin
   case TramaRx.Tipo of
    TRAMA_INFO:
      begin
      RecepcionCorrecta := True;
      EnviarTrama (TRAMA_ACK);
      end
    else
      HaHabidoUnError := True;
   end; { case }
  end { CRC correcto }
  else
   EnviarTrama (TRAMA_NACK);
```

Se ha definido un procedimiento para enviar las tramas que hemos declarado como EnviarTrama y que toma como único parámetro el tipo de la trama que queremos enviar. Con ello, conseguimos emplear un único procedimiento para realizar el envío de todas las tramas posibles (TRAMA_ACK, TRAMA_NACK y TRAMA_INFO). La implementación de este procedimiento la veremos unas líneas más adelante (se puede ver en la Figura 1).

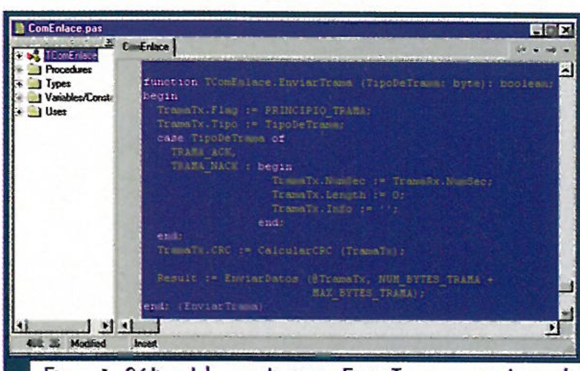


Figura 1.- Código del procedimiento *EnviarTrama* mostrado en el entorno de desarrollo de *Delphi*.

En el código, se puede destacar la presencia de dos variables *booleanas* - **RecepcionCorrecta** *y* **Ha HabidoUnError** -, que nos permitirán conocer, respectivamente, si se ha recibido correctamente una trama con información y si ha habido un error, producido por un imprevisto en el protocolo.

Utilizaremos variables booleanas para saber si la transmisión o la recepción han sido correctas

En cuanto al código que desarrolla todos los casos contemplados cuando el estado del componente es **etEsperaDatos**, es algo más complejo, pues los casos posibles aumentan:

```
etEsperaAck:
 if ComprobarCRC (---) then begin
  case TramaRx.Tipo of
   TRAMA_ACK:
     begin
     Timeout.Enabled:= False;
     TransmisionCorrecta:= True;
     estadoTx:= etEsperaDatos;
     end;
   TRAMA_NACK:
     begin
     Timeout.Enabled := False;
     ReenviarUltimaTrama;
     end;
   TRAMA_INFO:
     begin
     estadoTx := etEsperaDatos;
     Timeout.Enabled := False;
     Randomize;
     Timeout.Interval :=
      Random(TIMEOUT_TX_VARIA-
    BLE);
     Timeout.Enabled := True;
     end;
   else
     HaHabidoUnError:= True;
  end; { case TramaRx.Tipo }
 end { CRC correcto }
 else begin
  Timeout.Enabled := False;
  TransmisionCorrecta := False;
  ReenviarUltimaTrama;
 end; { CRC erróneo }
end; { etEsperaAck }
```

En este caso hay que destacar la llamada a un nuevo procedimiento – **ReenviarUltimaTrama** – en el que el transmisor debe ser capaz de retransmitir la última trama que envió.

También aparece una nueva variable *booleana* - **HaHabidoUnError** - que recoge el valor *TRUE* cuando la transmisión de una trama de datos ha sido correcta (es decir cuando hemos recibido un *ACK*).

Esta variable, junto a las otras dos *booleanas* de antes, las inicializamos a *FALSE* al inicio del procedimiento **ActuarSobreTrama.**

Al final de este procedimiento, y de acuerdo con los valores de estas variables *booleanas*, realizaremos las llamadas correspondientes a los eventos que sirven para informar al nivel superior:

```
if HaHabidoUnError
and
Assigned (FOnComEnlaceError) then
   FOnComEnlaceError (Self,
     ord(UltimoEstadoTx));

if TransmisionCorrecta and
Assigned(FOnTxCorrect) then
   FOnTxCorrect (Self);
if RecepcionCorrecta and
Assigned (FOnInfoReceived) then
   FOnInfoReceived (Self, Tra-
     maRx.Info, TramaRx.Length)
```

Figura 2.- Diagrama temporal de representación del funcionamiento del protocolo.

Es decir, si ha habido un error imprevisto, informamos al nivel superior mediante un código (que en este caso corresponde al estado en que nos encontrábamos en el momento en que se produjo al error). Si ha llegado un *ACK* enviamos el evento **OnTxCorrect** y si hemos recibido datos correctamente enviamos el evento **OnInfoReceived.**

Si llega un ACK de la trama que hemos enviado, disparamos el evento OnTxCorrect

Con este código queda finalizado el procedimiento **ActuarSobreTrama** en el que, como hemos podido ver, se decide el siguiente paso a tomar en función del tipo de la trama recibida y de la información que contiene.

PROCEDIMIENTO DE ENVÍO DE TRAMAS

El envío de las tramas lo implementaremos mediante una función única que hemos denominado **EnviarTrama.** Éste, tendrá un sólo parámetro de entrada, el tipo de la trama que queremos enviar, y

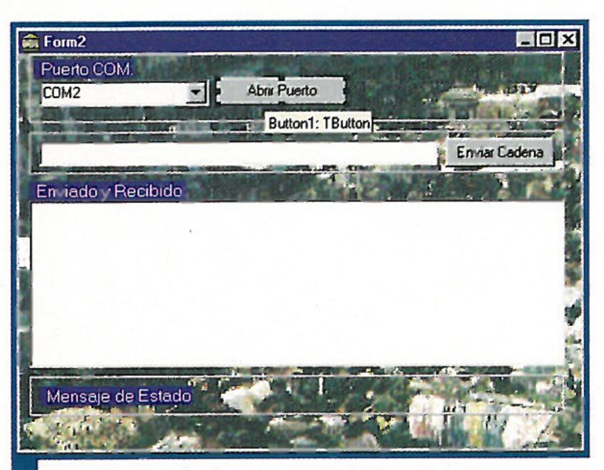


Figura 3.- Ejemplo de construcción del formulario *Form2.*

que en nuestro caso sólo podrá ser una de las tres tramas definidas:

```
TRAMA_ACK
TRAMA_NACK
TRAMA_INFO
```

El código de la función **EnviarTrama** (ver Figura 1) asignará a los campos de la estructura de datos que almacena la trama a transmitir **(TramaTx),** los valores correspondientes para cada tipo de trama.

Para transmitir la trama utilizamos la función de transmisión de datos del nivel 1

Si la trama es un *ACK* o un *NACK*, en este procedimiento podremos asignar los valores de todos los campos de la estructura **TramaTx.** Sin embargo, para una trama que transporte información útil, los campos: **NumSec** (número de secuencia), **Length** (longitud de la trama) e **Info** (datos del nivel superior), habrá que asignarlos en un procedimiento específico, que aprovecharemos para definir como interfaz del nivel superior.

En **EnviarTrama,** el campo *CRC* lo obtenemos de la propia trama, a través de un procedimiento denominado **CalcularCRC** que implementa el cálculo de un código de redundancia cíclico de 16 *bits* generado mediante un polinomio de acuerdo a la norma de la **UIT-T** (cada *bit* del código se genera mediante un polinomio formado por todos los bits de la trama). El código completo del procedimiento **CalcularCRC** se puede ver, junto con el resto del componente, en el *CD-ROM* de la revista.

CalcularCRC devuelve una variable de tipo word que contiene los 16 bits del CRC obtenido a través de los bits de la trama. Por su parte, la función **ComprobarCRC** cuya llamada podemos ver en el primer fragmento de código de este artículo, es muy sencilla: sólo tiene que llamar a **CalcularCRC** y comparar el resultado con el *CRC* que ha llegado en la trama:

```
function TComEnlace.ComprobarCRC
    (ATrama: TTrama; ACRC: word):
    boolean;
begin
 if ACRC=CalcularCRC(ATrama) then
   Result := True
 else
   Result := False;
end; {ComprobarCRC}
```

Una vez que tenemos todos los campos de la trama, sólo queda enviarla, para lo cual utilizamos la función del nivel 1 **EnviarDatos** (que proporciona el componente **ComSerie**). A esta función, le pasamos como parámetros la dirección de la estructura de datos de la trama (**@Trama**) y el tamaño de la estructura completa; obtenemos como resultado un *boolean* que informa del éxito o fracaso del envío. Ese mismo *boolean* será el valor devuelto en nuestra función.

ENVÍO DE TRAMAS DE DATOS

Las tramas de datos, aunque hacen uso de la función **EnviarTrama** para rellenar algunos datos de la trama y realizar el envío, hacen uso de una función específica para proceder a su envío. La razón es que esta función es la que utilizará el nivel superior para enviar las tramas con información útil, y por tanto estará definida en la parte *public* de la clase (a diferencia de **EnviarTrama** cuya declaración irá en la parte *private*).

La función que permite transmitir tramas con datos, servirá de interfaz con el nivel superior

El nombre que hemos dado a esta función es **EnviarTramaInfo** y su código completo se muestra en la Figura 3.

En este procedimiento, comprobamos antes de nada si el tamaño de la información a transmitir es mayor que el máximo tamaño de la trama y en caso afirmativo no continuamos y devolvemos un código de error (mediante el evento **FOnComEn laceError**). La razón de este proceder es que entre las funciones del nivel de enlace no se encuentra la segmentación de los datos, por lo

que corresponde al nivel superior entregar la información debidamente segmentada en el tamaño adecuado a la trama del nivel de enlace.

También generaremos un evento **FOnComEnlaceError** cuando, estando en el estado de espera de *ACK*, el nivel superior trata de enviar una nueva trama. Esto es debido a que el protocolo está implementado como "parada y espera" y por tanto, no permite enviar nuevas tramas hasta que no ha llegado la confirmación de la última trama enviada.

No podremos enviar una trama si estamos esperando un ACK: el protocolo es de "parada y espera"

En el caso en que todo sea correcto, rellenamos los campos de la estructura **TramaTx**. El campo **Info** almacenará la información que quiere transmitir el nivel superior. El campo **Length** almacenará la longitud de esa información. El campo **NumSec** tendrá el valor 0 ó 1, de forma secuencial. Por último, el resto de los campos se rellenarán en la función **En viarTrama** a la que se llama al final de ésta.

Antes de proceder al envío, reseteamos el *timeout* y fijamos el estado de transmisión en **etEs peraACK**.

Figura 5.- Aplicación *chat* de prueba, en el estado de transmisión.

REENVÍO DE TRAMAS

Cuando, después de haber enviado una trama de datos, recibimos una trama *NACK* o cuando vence un *timeout*, debemos reenviar automáticamente (y de forma transparente al nivel superior) la última trama enviada. Esto lo realizaremos en un procedimiento que hemos denominado **ReenviarUltimaTrama.**

Dado que reenviamos la última trama enviada, no debemos preocuparnos de volver a rellenar la estructura de datos **TramaTx**, ya que contendrá los datos correctos. Tan sólo habrá que llamar a la función **EnviarTrama** pasando como parámetro **TRAMA _INFO**.

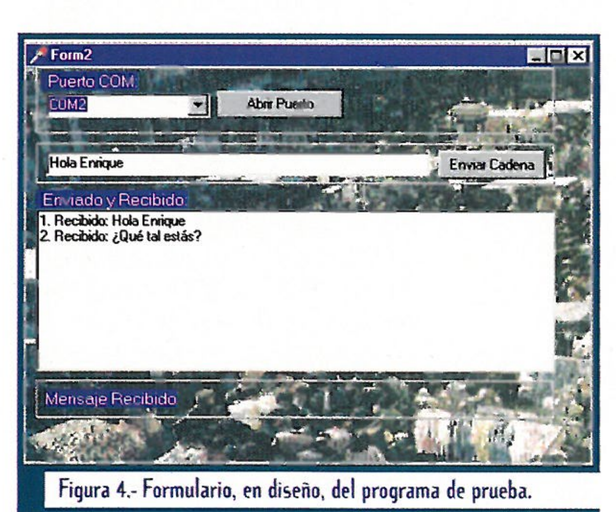


Figura 4.- Formulario, en diseño, del programa de prueba.

Sin embargo, sí deberemos ocuparnos de saber cuántas veces hemos reenviado la última trama, para proceder a la finalización de la transmisión al llegar a **NUM_MAXIMO_REPETICIONES**. Para controlar este extremo definimos una variable estática que almacene el último número de secuencia de la trama enviada; de esta manera, bastará comprobar si la trama que se pretende reenviar tiene el mismo número de secuencia que la que retransmitimos por última vez. En caso afirmativo, y a través de otra variable estática, almacenamos el número de reintentos. La definición de estas dos variables estáticas es la siguiente:

```
const NumRepeticiones: integer=
    0;
const PrevNumSec: integer = -1;
```

La variable **PrevNumSec** comenzará valiendo –1, para estar seguros de que la primera vez que entremos en el procedimiento el número de secuencia no será igual al anterior (que no ha habido ninguno).

Para controlar el número de reintentos almacenamos el último número de secuencia

Si llegamos a que **NumRepeticiones** llega a valer **NUM_MAXIMO_REPETICIONES**, generare-

mos el evento **OnBrokenLine**, para avisar al nivel superior de que el receptor no contesta.

PROGRAMA DE PRUEBA

El programa de prueba, cuyo formulario se muestra en la Figura 5, será prácticamente igual al que en su momento diseñamos para probar el componente **TComSerie**. Las únicas diferencias vendrán dadas, por la llamada a la función de transmisión de datos y por los manejadores de eventos.

En concreto, habrá que definir manejadores para los eventos: **OnInfoReceived**, **OnTxCorrect**, **OnBrokenLine** y **OnComEnlaceError**. Los nombres elegidos para estos procedimientos son, respectivamente: InfoRecibida, TxCorrecta, LineaRota y ErrorEnlace. El código correspondiente, como se puede ver en el Listado 1, sólo muestra en pantalla, a modo informativo, qué es lo que está pasando "por debajo", en el nivel de enlace.

En un programa de aplicación real, habría que utilizar estos eventos, para realizar de forma automática el procesamiento necesario ante cualquiera de ellos.

Por cierto, para comprobar que el protocolo funciona, si desconectamos el cable y enviamos una cadena de texto (es decir, una trama), al volver a conectar el cable, el receptor automáticamente buscará el comienzo de una trama y detectará correctamente alguno de los reenvíos.

Listado 1. Código que ha cambiado en el programa de prueba.

```
procedure TForm2.FormCreate(Sender: TObject);
begin
...
  MiComEnlace.OnInfoReceived := InfoRecibida;
  MiComEnlace.OnTxCorrect := TxCorrecta;
  MiComEnlace.OnBrokenLine := LineaRota;
  MiComEnlace.OnComEnlaceError := ErrorEnlace;
end;

...

procedure TForm2.Button2Click(Sender: TObject);
begin
  if MiComEnlace.EnviarTramaInfo (pChar(Edit2.Text), Length(Edit2.Text)) then ...
end

procedure TForm2.InfoRecibida (Sender: TObject; Informacion: pChar; numBytes: integer);
begin
  inc(cont);
  Memo1.Lines.Add (IntToStr(cont) + '. Recibido: ' + string(Informacion));
  Label2.Caption := 'Mensaje Recibido';
end;

procedure TForm2.TxCorrecta (Sender: TObject);
begin
  inc(cont);
  Memo1.Lines.Add (IntToStr(cont) + '.  *** Transmisión Correcta');
  Label2.Caption := 'ACK Recibido';
end;

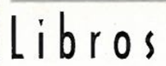
procedure TForm2.LineaRota (Sender: TObject);
begin
  inc(cont);
  Memo1.Lines.Add (IntToStr(cont) + '.  *** El Receptor está apagado o la Línea rota');
  Label2.Caption := 'Fin de la comunicación';
end;

procedure TForm2.ErrorEnlace (Sender: TObject; tipo: integer);
begin
  inc(cont);
  Memo1.Lines.Add (IntToStr(cont) + '.  **** ERROR Nº ' + string(tipo));
  Label2.Caption := 'Ha habido un error en el protocolo';
end;
```

CONCLUSIÓN

Con este artículo hemos terminado el desarrollo del componente *Delphi* que implementa el nivel de enlace y hemos utilizado, con pequeñas modificaciones, la misma aplicación de prueba que empleamos en su momento para comprobar el correcto funcionamiento del nivel físico (un *chat* a través del puerto serie).

A partir de aquí, el desarrollo debería continuar en un componente, heredado de **TComEnlace** que realizase las funciones propias del nivel de aplicación, como por ejemplo segmentar los mensajes mayores que el tamaño máximo de la trama, o trocear un archivo en fragmentos, de forma que pudiesen añadirse anexos a las transmisiones de mensaje de texto, etc.

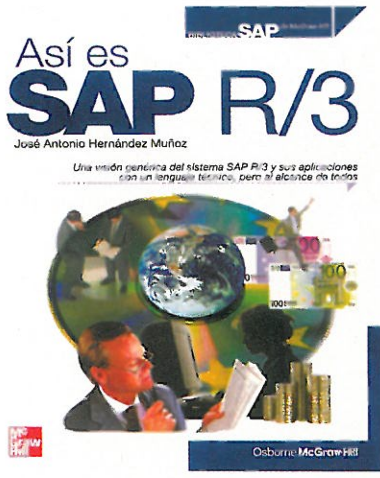




ASÍ ES SAP R/3

SAP R/3 se conoce comúnmente como el sistema estándar de aplicaciones de gestión que cubre las necesidades críticas y de información de las empresas. Tanto *SAP* como su producto *R/3* se han convertido en uno de los mayores éxitos de la industria informática de los último años.

Este libro le presenta al lector una visión general del sistema *SAP R/3,* que abarca desde las características del *software* a las metodologías de implantación. Trata con un lenguaje claro y sencillo todos los contenidos, y además proporciona a los consultores un excelente manual de referencia para conocer este sistema de un modo global.



Las páginas de este manual ofrecen una gran cantidad de consejos prácticos para saber moverse por el sistema desde el punto de vista del usuario.

Algunos de los contenidos que muestra son: guía introductoria al *SAP R/3*, visión general de los módulos funcionales, la implantación, formación genérica del sistema, *Bussines Workflow, Archiving,* sistema de autorizaciones, arquitectura *SAP*, o entorno de programación *ABAP* entre otros.

Editorial: Mc Graw Hill	Autor: José A. Hernández Muñoz
Nº de Páginas: 253	Idioma: Español
Nivel: Avanzado	Precio: 3.500 Ptas. (I.V.A. inc.)

NÚCLEO DEL API WIN 32

Este ejemplar ofrece a los desarrolladores una guía completa y detallada de la *interfaz* de la programación de aplicaciones de *Microsoft Windows*, pieza básica del desarrollo para esa plataforma. Esta referencia cubre las funciones más comunes del *API* de *Windows*, desde la creación de ventanas y las funciones de gestión de memoria hasta las funciones del manejo del portapapeles y tratamiento de errores.

La documentación de cada función incluye sintaxis, una descripción del objetivo de la función, la presentación detallada de los parámetros y sus posibles valores, referencias a otras funciones relacionadas con ella y un ejemplo de su utilización en *Object Pascal*, el lenguaje de programación de *Delphi.*

Si usted es un programador de *Delphi* con conocimientos básicos de la programación para *Windows*, este libro le mostrará lo que el *API* de *Windows* puede hacer por usted, y le ayudará a producir aplicaciones más eficientes e impactantes, y a extender la funcionalidad de sus componentes.

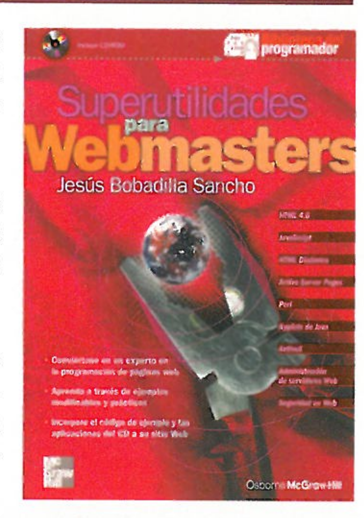
Editorial: Danysoft Internacional	Autor: John Ayres y otros
Nº de Páginas: 797	Idioma: Español
Nivel: Intermedio-Avanzado	Precio: 7.900 Ptas. (I.V.A. inc.)

SUPERUTILIDADES PARA WEBMASTERS

Este libro es una herramienta imprescindible para todas las personas relacionadas con la creación de aplicaciones *Web*, o que quieran aprender a crearlas. Se tratan varias tecnologías y lenguajes, y se proporcionan utilidades que pueden ser incorporadas en los sitios *Web* que se estén desarrollando.

Va dirigido a programadores de páginas *Web*, analistas y diseñadores de sitios *Web*, *Webmasters* y personas con conocimientos de programación. Los lenguajes se tratan de tal forma que si el lector asimila los conceptos explicados en los ejemplos de programación se puede considerar un experto en creación de aplicaciones *Web*.

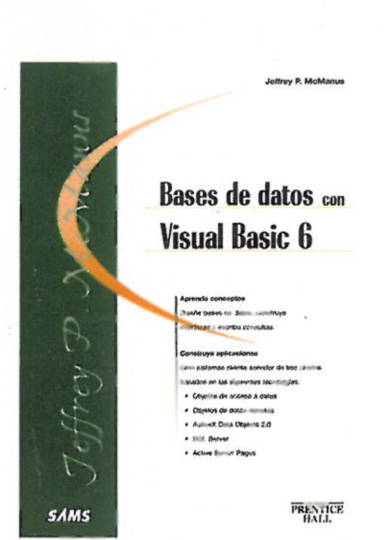
A pesar de que esta publicación abarca diversas áreas, proporciona un nivel alto de especialización en las materias tratadas, de este modo se ofrece una gran cantidad de código de "superutilidades" que, de forma directa o modificada, se pueden incorporar a sus aplicaciones, tales como: *chat* con varios canales, menús encadenados, servicio *FTP*, acceso a bases de datos vía *ODBC*, generación de gráficos, etc.



Editorial: Mc Graw Hill	Autor: Jesús Bobadilla Sancho
Nº de Páginas: 695	Idioma: Español
Nivel: Intermedio-Avanzado	Precio: 6.500 Ptas. (I.V.A. inc.)

BASES DE DATOS CON VISUAL BASIC 6

Aprenda las mejores técnicas y trucos del experimentado autor *Jeffrey McManus*. Emplee estas aplicaciones, soluciones y proyectos del mundo real en sus propios programas y conviértase en el programador más eficiente en acceso a bases de datos con *Visual Basic* 6. Este libro le ofrecerá todo lo necesario para escalar el siguiente peldaño del acceso a bases de datos con la programación en *Visual Basic* 6.



Aborda todas las tecnologías de acceso a bases de datos que están disponibles para el desarrollador de *VB*, así como algunas que no lo están del todo para resaltar los aspectos importantes de éstas.

Gracias al libro llegará a configurar y diseñar sistemas cliente-servidor, crear componentes de *software* conducido por bases de datos, desplegar soluciones de bases de datos en *LAN* corporativas o en *Internet*, trabajar con *AvtiveX Data Objects* 2.0

El manual viene acompañado con un *CD-ROM* que incluye todo el código fuente del autor, así como decenas de aplicaciones y ejemplos en *Visual Basic*.

Editorial: Prentice Hall	Autor: Jeffrey P. McManus
Nº de Páginas: 742	Idioma: Español
Nivel: Intermedio-Avanzado	Precio: 6.500 Ptas. (I.V.A. inc.)

Dudas técnicas

En esta sección, como cada mes, SÓLO PROGRAMADORES os brinda la oportunidad de encontrar respuesta a las dudas que podáis tener, en cualquier tema relacionado con la programación y bajo cualquier entorno de desarrollo. Ya sabéis que nuestra dirección es solop@virtualsw.es

PREGUNTA

Hola gente de *Sólo Programadores:*

Lo primero de todo es felicitarles por su trabajo. Su revista se está convirtiendo en el punto de referencia para todo aquel que quiera estar al día en ordenadores en este país. En cualquier caso, hace tiempo que vi en su revista alguna imagen de un ordenador en el que se veía la televisión.

La verdad es que me sería enormemente útil algo así, ya que vivo con mis padres y, además del monitor del *PC* que necesito para mis estudios de informática, no me cabe una tele. Por lo tanto la pregunta que tengo es la siguiente:

¿Es posible ver la televisión en el *PC* de casa? Cuando digo televisión, me refiero a la emisión propiamente dicha o a la señal que viene de un vídeo *VHS*, y no a ver películas de las que venden en *DVD*. Si es posible, ¿es necesario escribir algún tipo de programa para conseguirlo? ¿En qué lenguaje?

Finalmente: ¿Es posible hacer otra cosa con el *PC* al tiempo que se ve la televisión? Ésta sale en una ventana o ¿es a pantalla completa?

Yo tengo *Windows 98* en el *PC*, aunque también he instalado *Linux* (aunque no lo uso mucho), así que me valdría cualquiera de las dos opciones.

Muchas gracias por su respuesta y, si pudieran incluir precios aproximados, se lo agradecería enormemente. Gracias de nuevo.

I.P.G. Madrid.

RESPUESTA

Estimado lector:

La respuesta a tu primera pregunta es muy sencilla: sí, es posible ver la televisión en casa, utilizando el *PC* como si de un aparato de televisión cualquiera se tratara.

Por lo tanto, utilizando el *hardware* apropiado, podrás acceder a las cadenas públicas, privadas, así como a las emisoras que se reciben por la parabólica - si la hubiera en tu casa, - o los reproductores de videocassette *VHS*.

Para conseguirlo sólo hace falta tener un *PC* relativamente moderno, una tarjeta sintonizadora de televisión y una tarjeta de sonido. Las características mínimas de un *PC* en el que nos conste que se ha utilizado una tarjeta de televisión con resultado óptimos es:

- Procesador *AMD K6-200 Mhz.*
- 32 *Mb* de *RAM.*
- Tarjeta de Vídeo *S3 ViRGE PCI* con 2 *Mb.*
- Tarjeta de Sonido *SoundBlaster Pro.*
- Monitor Digital 14".

En cuanto al modelo de la tarjeta que podrías utilizar para ver la televisión en tu ordenador personal, existen numerosas posibilidades. La que nosotros conocemos y, hasta ahora, ha funcionado estupendamente es la *AverMedia TVCapture98* cuyo precio en el mercado debe rondar las 15.000 pesetas.

Normalmente no es necesario escribir ningún tipo de programa para poder utilizar este tipo de accesorios en nuestro *PC*. Todas las tarjetas de este estilo traen *drivers* y aplicaciones para ser utilizadas en *Windows 98*. En cuanto

a *Linux*, si la tarjeta de captura está basada en el *chip* Bt848/878 el propio *Kernel* del sistema incluye soporte para la misma. En caso contrario, escribir un *driver* requerirá una descripción a nivel de registro de la tarjeta, unos profundos conocimientos de *C* y ensamblador, así como una idea clara de los interfaces *IOCTL* de *Unix*.

La utilización de la *CPU* que hacen este tipo de tarjetas es prácticamente negligible, por lo que el ordenador podrá estar realizando otras tareas al mismo tiempo que reproduce las imágenes en la pantalla.

En cuanto al formato de salida, todas las sintonizadoras de televisión que han pasado por nuestras manos en los últimos años permiten, tanto el formato de pantalla completa como el de ventana. En cualquier caso, te aconsejamos que consultes a tu proveedor habitual para que te informe de los detalles específicos de cada modelo que te ofrezca.

PREGUNTA

Hola *Sólo Programadores!!!*

Hace tiempo que estoy realizando una pequeña aplicación para un pequeño *gadget* cuyo corazón es un ordenador basado en un antiguo microprocesador de *Motorola*: el 6809. Al principio pensé que sería similar al 68000 del que poseo grandes cantidades de información. Sin embargo, después de mucho probar, y después de analizar algo del código que había en una de las *ROM's* del *gadget*, vi que era totalmente diferente.

Como podéis imaginar, el proceso de programación del aparato es muy complejo. Primero escribo en papel el código máquina correspondiente a la parte del programa que quiero introducir. A continuación lo verifico sobre el mismo papel y finalmente lo introduzco en la *RAM* a través de un pequeño *panel frontal* instrucción a instrucción.



Figura 1.- Gracias a los sintonizadores de televisión, no sólo podemos ver la TV, sino además capturar nuestras escenas preferidas.

Como veis es un absoluto engendro. Para probar cualquier cosa tengo que introducir todo el programa, *byte* a *byte* y para cualquier operación compleja: multiplicación, impresión, etc..., me tengo que fabricar la rutina que lo haga.

Aunque sea muy antiguo, ¿podríais darme alguna información sobre este microprocesador? ¿Conocéis algún programa para *PC* que me permita escribir código para el 6809? ¿Sabéis de alguna rutina rápida para multiplicar dos números arbitrarios en él? ¿Y para multiplicar por números fijos como por ejemplo potencias de dos?

Muchas gracias por vuestras respuestas. Sé que puedo confiar en vosotros.

PS: Aunque preferiría un programa para *PC*, también tengo un *iMac* por lo que si el programa para 6809 estuviera disponible para *MacOS*, también me sería muy útil.

RESPUESTA

Estimado lector:

Hacía años que no oíamos hablar del 6809. Con la creación de este microprocesador se llegó a la cúspide en el diseño de *chips* de 8-*bits* en la época dorada de los microordenadores a finales de los 70 y principios de los 80.

Del 6809 se comercializaron, - si nuestra memoria no nos falla, - dos versiones: 6809 y 6809e, cuya única diferencia consistía en que el 6809e admitía relojes externos en vez de generarlos *on-chip*. El motivo fundamental de esta variación era debido a la necesidad de que un *chip SAM* (*Sequential Address Multiplexor*) pudiera marcar la temporización.

Sin lugar a dudas el 6809, - en ambas versiones, - fue el *chip* de 8 *bits* más elegante jamás diseñado. Entre sus características arquitecturales podemos incluir:

- 4 registros índice de 16 *bits: X, Y, U* y *S*. Los dos últimos servían como pila de usuario y del sistema respectivamente.
- 2 acumuladores de 8 *bits: A* y *B*.
- 1 acumulador de 16 *bits: D*, compuesto por *A* y *B*.
- Velocidad de Reloj: 0.86 *Mhz*, modificable por *software*.
- Multiplicación *hardware*.
- Hasta 64 *Kb*. de *RAM* (*16-bit* en el *bus* de direcciones).

Además, el 6809 permitía la creación de código que fuera independiente de su posición en la memoria *RAM*. Todos los datos podían

ser accedidos relativos al registro contador de programa, lo cual permitía la independencia de posición sin necesidad de ninguna sobrecarga del monitor de ejecución.

Entre los ordenadores basados en este *chip*, se incluye el primer microordenador fabricado en España: el *Dragón*. Una magnífica máquina en sus tres versiones, - *Dragon* 32, *Dragon* 64 y *Dragon* 200, - cuyo éxito fue relativo si lo comparamos con su contemporáneo el *ZX Spectrum*, fundamentalmente por la mayor cantidad de juegos disponibles para éste.

Para solucionar tu problema, o bien facilitar el proceso de programación del *gadget* que has de manejar, se nos ocurren dos opciones:

La primera de ellas consiste en adquirir un viejo *Dragon 32* y realizar las pruebas en él. Si vas a utilizar este procedimiento te aconsejamos que te hagas con una copia del *Ensamblador/Monitor AllDream*, herramienta imprescindible para crear programas en ensamblador con esta máquina.

La segunda es utilizar alguno de los emuladores de *Dragon 32/64* disponibles en la Red. Es especialmente bueno el llamado "*T3 Dragon/CoCo Emulator*" creado por *Paul Burgin.* Gracias a este emulador es posible realizar programas en *Ensamblador* del 6809 con la comodidad de poseer un disco duro, un monitor en condiciones, etc...

En cuanto a tus preguntas relativas a la multiplicación, vamos a intentar responderlas lo más exactamente posible.

Si lo que se pretende es multiplicar dos valores, - de 8 *bits*, - entre sí y acceder al resultado, - que puede llegar a tener hasta 16 *bits,* - no es necesario realizar ningún tipo de rutina a tal efecto. El 6809 fue, probablemente, el primer procesador de gran consumo en poseer una instrucción de multiplicación hardware: *MUL*.

El proceso sería el siguiente:

1.Cargamos los diferentes valores en los acumuladores *A* y *B*.
2.Utilizamos la instrucción *MUL*.
3.Colocamos el valor del acumulador compuesto *D* en la posición de memoria requerida.

Si escribimos el código asumiendo 16 y 2 como valores a multiplicar, podríamos tener:

```
7F01   LDA #$108610
7F03LDB #$02C602
7F05LDX #$60008E6000
7F08MUL3D
7F09STD 0,X+ED80
```

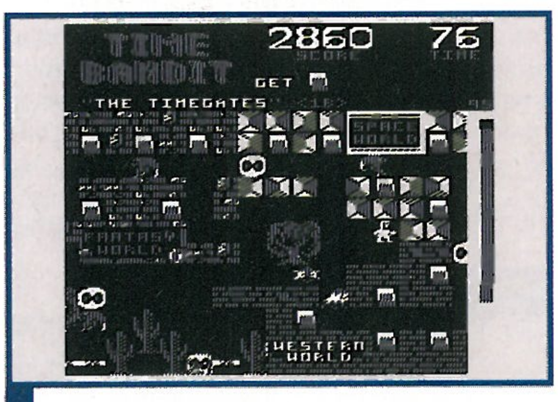


Figura 2.- Entre los magníficos programas disponibles para el *Dragon 32* se encontraban juegos de la calidad del *Time Bandit.*

Esta multiplicación es relativamente rápida y muy sencilla de programar. Sin embargo, para el caso de que uno de los multiplicandos sea una potencia de dos, lo que se puede hacer es utilizar las instrucciones de decalaje y rotación para emular la multiplicación. Si, por ejemplo, quisiéramos multiplicar el número 53 por 2, para obtener 106, lo que haríamos sería lo siguiente:

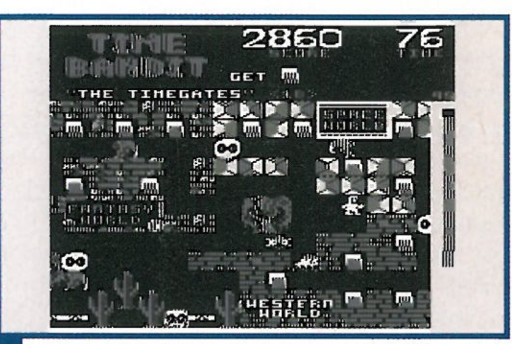


Figura 3.- El 6809 fue utilizado en el primer ordenador de manufactura española: el *Dragón.*

```
7F01   LDA #$358635
7F03LDX #$60008E6000
7F06ASLA48
7F07STA 0,X+A780
```

Una vez acabado el programa, tendríamos el valor 106 en la dirección de memoria *$6000.*

En cualquier caso, y para ayudarte en la interesante tarea de programar un 6809, te aconsejamos que te hagas con una copia del libro:

LENGUAJE MÁQUINA
DEL DRAGÓN
Ian Sinclair
Editorial Gustavo Gili S.A.
Barcelona 1985
ISBN 84-252-1243-X

O bien la versión inglesa del mismo:

Introducing Dragon
Machine Code
Ian Sinclair
Granada Publishing 1984

En cualquier caso, desde *Sólo Programadores* quedamos a tu disposición para cualquier consulta posterior.